CDC 00594

The Transactor

The Tech/News Journal For Commodore Computers

95% Advertising Free! Jan. 1986: Volume 6, Issue 04. $2.95

Implementing The Sciences

- The Projector For The 64: Plot In 3 Dimensions
- Sky Travel: The Astronomers' Accessory Reviewed
- Test Report: The 6581 SID Programmable Filter
- Projectile Motion: Path Calculating With Mechanics
- The Unassembler: Make Real Source Code Out Of Object Files
- The Compressor: Squeeze Files By Up To 50%
- Solving SAVE@: The Search Continues
- Gaussian Elimination and Calculating Accurate Sums
- Even More Indestructible Variables
- Microsecond Timer For The 64
- SID Super Sound

Plus:

Notes on the 1541 Reset Sequence
and the New 1571 Floppy Drive

Carlo Mostacci '85

0 72246 00594 01

The Transactor

Disk Has Arrived!

Simply code your co-ordinates onto the postage powered order form and every program from each issue will be locked in, energized, and transported from our starbase directly to yours! Warp 9 will seem slow compared to the time you save typing, and the programs will give your machine that look and feel of a fresh set of Dilithium Crystals! Coast through the Neutral Zone with The Transactor Disk!

Only $7.95 Each!
6 Disk Subscription
Just $45.00!

J. Mostacci

Volume 6
Issue 04
Circulation 64,000

Implementing The Sciences

Note: Before entering programs, see "Verifizer" on page 4

The Transactor
The Tech/News Journal For Commodore Computers

Managing Editor
Karl J. H. Hildon

Editor
Richard Evers

Technical Editor
Chris Zamara

Art Director
John Mostacci

Administration & Subscriptions
Lana Humphries

Contributing Writers
Ian Adam
Gary Anderson
Daniel Bingamon
Anthony Bryant
Tim Buist
Jim Butterfield
Gary Cobb
Bob Davis
Elizabeth Deal
Tony Doty
Michael J. Erskine
Jim Grubbs
Dave Gzik
Tom Hall
Bob Hayes
John Jay Hilfiger
Jesse Knight
Jack Lothian
Scott Maclean
Jim McLaughlin
Gerald Neufeld
Noel Nyman
Richard Perrit
Glen Reesor
John W. Ross
Louis F. Sander
Edward Smeda
Darren J. Spruyt
Nick Sullivan
Zoltan Szepesi
Karel Vander Lugt
Audrys Vilkas
Jack Weaver
Charles Whittern

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix–ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') has a flat top as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print " flush right " – would be shown as – print "[10 spaces]flush right "

Cursor Characters For PET / CBM / VIC / 64

Down – [q]
Up – [Q]
Right – [ ]
Left – [Lft]
RVS – [r]
RVS Off – [R]
Insert – [ ]
Delete – [t]
Clear Scrn – [ ]
Home – [s]
STOP – [c]

Colour Characters For VIC / 64

Black – [P]
White – [e]
Red – [£]
Cyan – [Cyn]
Purple – [Pur]
Green – [↑]
Blue – [←]
Yellow – [Yel]
Orange – [A]
Brown – [U]
Lt. Red – [V]
Grey 1 – [W]
Grey 2 – [X]
Lt. Green – [Y]
Lt. Blue – [Z]
Grey 3 – [Gr3]

Function Keys For VIC / 64

F1 – [E]
F2 – [I]
F3 – [F]
F4 – [J]
F5 – [G]
F6 – [K]
F7 – [H]
F8 – [L]

Please Note: The Transactor has a new phone number: (416) 878 8438

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725–050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209 **ISSN# 0827-2530.**

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 878 8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

Back Issues: $4.50 each. Order all back issues from Milton HQ.

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 04, 05, 06, Vol 5 Issues 03, 04
Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 04, 05, 06. Vol. 6: 01, 02, 03, 04

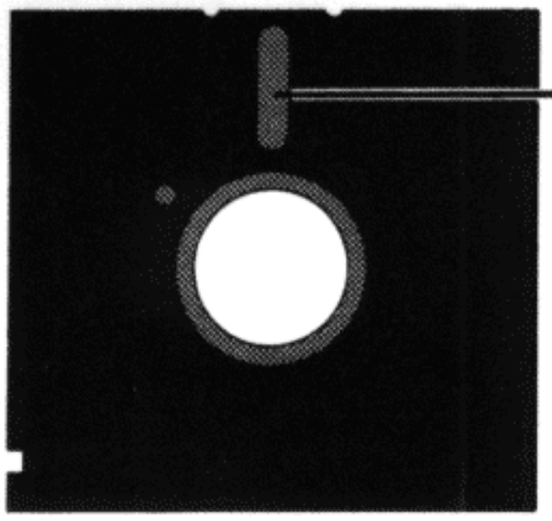
Quantity Orders:

CompuLit
PO Box 352
Port Coquitlam, BC
V5C 4K6
604 941 7911

U.S.A. Distributor:

Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(or your local wholesaler)



Start Address

The Amiga. It's been billed as "Commodore's Everything Machine?" and "The Ultimate Micro?" but I can think of only one word to describe it; Stunning!

In my last *Start Address*, my comments were somewhat less flattering. Commodore doesn't often come tracking us down for a "show and tell". In fact I still haven't seen one up close. What I have seen is an edited video of the official launch held at Lincoln Center in New York. Commodore's spending curbs were but a myth at this show.

There isn't a magazine rack in the world without an Amiga shown prominently under the computer titles. But reading any one of them won't have the impact of an audio-visual. The flick left no doubts about the phenomenal speed capabilities. High resolution graphics will impress just about anyone, until you start them moving. Not on the Amiga. Part of the video started with a ballerina on the Amiga screen. The program began with a stick drawing which eventually became a rather nicely coloured hi-res pic. Enter the real ballerina from stage right. The crowd liked how identical the two appeared. But when they both started twirling in synchro, the question in everyone's mind must have been "who's leading?". And I'm not so sure the Amiga wasn't taking it easy on 'er.

If you haven't seen one yet, don't pass up the opportunity for a demo. This machine will BLOW YOU AWAY. Even more impressive is the amount of effort behind some of the demos. Mere ten line programs create some of the most awesome displays imaginable. Memory is expandable to a whopping 8 Meg! With that kind of space to play in, I'm sure the best dazzlers have yet to be conceived.

PCophytes will find a new contender on the ballot. Yes, the Amiga will be PC compatible. A Lotus 1-2-3 production line diskette was no apparent struggle for the 68000 based machine. Commodore claims even the Sub-Logic Flight Simulator will port to the Amiga, but go on to say "why bother, an Amiga Flight Simulator is due shortly from Sub-Logic". My guess has this program as the first to go beyond the awesome demo.

Sound was equally impressive, although I think it will take more than a demonstration to tax the analog department. Speech synthesis appears to be included with the package, as well as sound digitizing. With a microphone, one can record any sound for future playback, and in stereo finally. The show included a short jam session with an Amiga connected to a keyboard, but I'm sure that combo also has a long road ahead of it.

Less visible (audible) is the fact that most of these tasks are performed with very little effort from the 68000. Three VLSI super chips handle operations that might otherwise take a good chunk of processor attention. This leaves the CPU plenty of time to move data around, and I get the impression these chips get awfully hungry.

I could go on for pages about 4,096 colours, DMA 880K microfloppies, I/O and expansion ports, ICON control, windowing, multitasking, etc., but I only have one. Future Transactors will spend time on the details but not until a few more get sold. Unofficially I heard that 5,000 are ready to be shipped, but where and to whom I don't know.

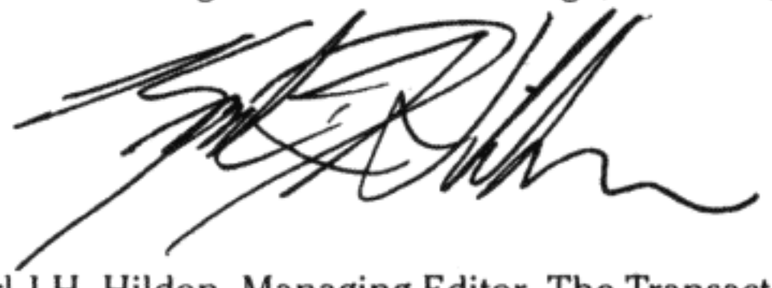
Amiga falls short of "the new wave" by my definition. However, if the Amiga doesn't stir up white water on Commodore shores, it's hard to imagine what will. Commodore has some PR patching to do with many a retailer that are not eager to add shelf space for any new lines, let alone CBM. Perhaps the calibre of this machine will help Commodore regain the healthy dealer relationship they'll need to attain success the Amiga deserves.

Aside from hardware, there's another great deal you should be aware of. Viewtron is a NAPLPS videotex service out of Miami, Florida. It's run by Knight-Ridder Newspapers Inc., a company with a mere 1.7 billion (yes Billion) in sales last year. However, Viewtron is by no means "new". For the last six years, Knight-Ridder has been developing this service to the tune of 40 million dollars. Originally it was available only to those willing to spend $600 on a videotex terminal, and only in south Florida. Special videotex software for the 64 is required to obtain some lovely picture graphics, and for just $9.95 it's yours. It comes with a perfect little manual and though I can't detail all the services Viewtron offers, the list is long and well documented. Once you get your software, you answer a few questions and it does the rest. Viewtron is available now through most of the major networks. On your first call you'll be asked for a credit card number and you only pay for the time you're on - your first hour is FREE.

Don't have a modem? Viewtron sells them too. And check this out. Get the software and a 300 baud Westridge 6420 for just $49.95, or a 1200 baud Volksmodem 12 for $189.95. This kinda stuff normally goes in News BRK, but I rather like bearing good news myself. In my opinion, Viewtron is the safest money you can spend for your 64. And Viewtron guarantees it. Their number is 1 800 543 5500 Operator# 825 (305 674 1444 in Canada).

Lastly, I hope it won't be long before our on-line plans go into full swing. Viewtron has approached us several times and we're anxious too. More next issue, or see you on Viewtron! (We still have plans for Delphi too!)

There's nothing as constant as change, I remain,



Karl J.H. Hildon, Managing Editor, The Transactor

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two-letter code for each program line which you can check against the corresponding code in the program listing.

There are two versions of VERIFIZER on this page; one is for the PET, the other for the VIC or 64. Enter the applicable program and RUN it. If you get the message, "***** data error *****", re-check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 828 to enable the C64/VIC version (turn it off with SYS 831)
or SYS 634 to enable the PET version (turn it off with SYS 637)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing it means we've editted that line at the last minute which changes the report code. However, this will only happen occasionally and only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re-check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors (eg. POKE 52381,0 instead of POKE 53281,0), but ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm-start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

Listing 1a: VERIFIZER for C64 and VIC-20

```
KE 10 rem* data loader for "verifizer" *
JF 15 rem vic/64 version
LI 20 cs=0
BE 30 for i=828 to 958:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
FH 60 if cs<>14755 then print "***** data error *****":end
KP 70 rem sys 828
AF 80 end
IN 100 :
EC 1000 data 76, 74, 3,165,251,141, 2, 3,165
EP 1010 data 252,141, 3, 3, 96,173, 3, 3,201
OC 1020 data 3,240, 17,133,252,173, 2, 3,133
MN 1030 data 251,169, 99,141, 2, 3,169, 3,141
MG 1040 data 3, 3, 96,173,254, 1,133, 89,162
DM 1050 data 0,160, 0,189, 0, 2,240, 22,201
CA 1060 data 32,240, 15,133, 91,200,152, 41, 3
NG 1070 data 133, 90, 32,183, 3,198, 90, 16,249
OK 1080 data 232,208,229, 56, 32,240,255,169, 19
AN 1090 data 32,210,255,169, 18, 32,210,255,165
GH 1100 data 89, 41, 15, 24,105, 97, 32,210,255
JC 1110 data 165, 89, 74, 74, 74, 74, 24,105, 97
EP 1120 data 32,210,255,169,146, 32,210,255, 24
MH 1130 data 32,240,255,108,251, 0,165, 91, 24
BH 1140 data 101, 89,133, 89, 96
```

Listing 1b: PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI 10 rem* data loader for "verifizer 4.0" *
CF 15 rem pet version
LI 20 cs=0
HC 30 for i=634 to 754:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50 :
OG 60 if cs<>15580 then print "***** data error *****":end
JO 70 rem sys 634
AF 80 end
IN 100 :
ON 1000 data 76,138, 2,120,173,163, 2,133,144
IB 1010 data 173,164, 2,133,145, 88, 96,120,165
CK 1020 data 145,201, 2,240, 16,141,164, 2,165
EB 1030 data 144,141,163, 2,169,165,133,144,169
HE 1040 data 2,133,145, 88, 96, 85,228,165,217
OI 1050 data 201, 13,208, 62,165,167,208, 58,173
JB 1060 data 254, 1,133,251,162, 0,134,253,189
PA 1070 data 0, 2,168,201, 32,240, 15,230,253
HE 1080 data 165,253, 41, 3,133,254, 32,236, 2
EL 1090 data 198,254, 16,249,232,152,208,229,165
LA 1100 data 251, 41, 15, 24,105,193,141, 0,128
KI 1110 data 165,251, 74, 74, 74, 74, 24,105,193
EB 1120 data 141, 1,128,108,163, 2,152, 24,101
DM 1130 data 251,133,251, 96
```

Bits and Pieces

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits & Pieces column, we'll credit you in the column and send you a free one–year's subscription to The Transactor

Multiple Directory Pattern–Matching

Commodore's filename pattern–matching feature for disk directories is more powerful than many people are aware. One little–used ability is the use of multiple patterns in a directory listing. For example, you could get a list of all files on the disk in drive zero starting with either the letter "S" or the letter "D":

```
LOAD"$0:S*,0:D*",8
```

Up to five selective directories may be used in a single directory filename.

Corrupting RAMTAS Routine

Edward Smeda, Victoria, Australia

RAMTAS ($FF87) is a C–64 Kernal routine which, among other things, has the function of setting the top of memory pointer. This is done by non–destructively testing RAM until it finds a memory location which does not return the value written to it. This location, usually $A000, then becomes the top of memory. RAMTAS is part of the C–64 power–up routine ($FCE2).

Normally, no problems occur with this routine. However, if you have any machine code or other information stored in the RAM under BASIC ROM you will find that a hardware reset (reset button) or software cold–start (SYS 64738) will always corrupt the byte at $A000. This occurs because when RAMTAS tests $A000, it writes the RAM with $55 but, on reading, it reads the BASIC ROM instead and finds a different value. RAMTAS aborts at this point, leaving $55 in the RAM at $A000.

While this does not really qualify as a bug, programmers should be aware that it does occur and should make allowances. There are a number of ways around the problem, but the simplest is to avoid using location $A000 for program or data.

Editor's note: On the other hand, this "feature" can be used to check if a reset occurred since a program was last RUN

Where am I?

Noel Nyman, Seattle WA

Relocatable machine language programs are the easiest to use. Invariably some nifty routine from The Transactor sits in a spot needed for another part of your program. it would be best if authors made their code relocatable. This isn't always easy. JMPs within the code are usually necessary and to use JMP commands, absolute addresses are required.

However, if the code can find its own location in memory, the JMP addresses can be calculated regardless of where the user stuck the program.

The "Where am I?" routine below stores a reference to its beginning address before executing the main program. It uses a JSR to force the program counter (the address of the JSR instruction) to the stack, then retrieves the address.

```
JSR  $FFDE ;read real–time clock, or any harmless JSR
TSX
DEX
DEX
TXS        ;move the stack pointer to the stored address
PLA
STA  $FD   ;store high byte of address
PLA
STA  $FC   ;store low byte
  (main program)
```

The vector stored at $FC/$FD is the starting address of "Where am I?" plus two. By adding an offset to this value and using indirect JMPs, the program can be made totally relocatable.

QUAKE!!

This is another one of those lovely Transactor specials, frivolous but somehow worth typing in anyway. QUAKE!! will simulate the effect of a 6.0 on the Richter scale, or programming while using hallucinogenics. Good at parties or for practi-

cal jokes; amaze your friends! The BASIC loader below will generate the 191 bytes of machine code which unleashes "quake mode" – you'll still be able to program normally while the quake is occurring. Quake mode is activated with SYS 49152 and turned off with SYS 49155. Make sure you have plenty of air–sickness bags nearby!

```
AA  10 rem* data loader for "quake" *
DK  11 rem* transactor magazine '85 -cz
KJ  15 rem  save"@0:quake.bas",8
LI  20 cs=0
KF  30 for i=49152 to 49342:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FB  60 if cs<>16666 then print"!data error!":end
DD  70 sys 49152
EP  80 rem sys 49155 to stop
KF  90 end
IN  100 :
IH  1000 data  76,  49, 192,  76, 112, 192,   0,   0
DA  1010 data   1,   2,   3,   4,   5,   6,   7,   7
PB  1020 data   7,   7,   7,   6,   5,   4,   3,   2
BA  1030 data   1,   0,   0,   0,   4,   5,   6,   7
DD  1040 data   7,   7,   7,   6,   5,   4,   3,   2
FP  1050 data   1,   0,   0,   0,   0,   1,   2,   3
OC  1060 data   4, 120, 169,  88, 141,  20,   3, 169
FG  1070 data 192, 141,  21,   3, 169,   1, 141,  26
NC  1080 data 208, 169,   0, 141,  18, 208, 173,  17
MM  1090 data 208,  41, 119, 141,  17, 208, 173,  22
EJ  1100 data 208,  41, 247, 141,  22, 208,  88,  96
JH  1110 data 173,  25, 208,  41,   1, 240,  11, 169
EF  1120 data   1, 141,  25, 208,  32, 150, 192,  76
KO  1130 data  49, 234, 104, 168, 104, 170, 104,  64
BP  1140 data 120, 169, 128, 141,  26, 208, 169,  49
LH  1150 data 141,  20,   3, 169, 234, 141,  21,   3
MK  1160 data 173,  22, 208,  41, 240,   9,   8, 141
AK  1170 data  22, 208, 173,  17, 208,  41, 240,   9
PO  1180 data  11, 141,  17, 208,  88,  96, 174,   6
CN  1190 data 192, 173,  22, 208,  41, 248,  29,   7
DG  1200 data 192, 141,  22, 208, 173,  17, 208,  41
HF  1210 data 248,  29,  28, 192, 141,  17, 208, 238
PK  1220 data   6, 192, 173,   6, 192, 201,  21, 144
PG  1230 data   5, 169,   0, 141,   6, 192,  96
```

The Schizophrenic Sprite

The shape of any C64 sprite is completely determined by 63 bytes in memory. To change the shape of a sprite, the sprite definitions are usually kept static, and pointers are changed to point to definitions elsewhere in memory. What about doing the opposite – keeping the sprite pointer constant but changing the 63 bytes defining the sprite? What if a sprite definition occurs in screen memory? To find out, enter this short bit of code:

```
10 rem schizo-sprite, cz85
20 vic=53248: rem vic chip at $d000
30 poke vic,25 : poke vic+1,100
40 poke vic+21,1: poke vic+39,1
50 poke vic+23,1: poke vic+29,1
60 poke 2040,16
```

A double–sized white sprite appears, whose shape changes depending on the first 63 characters on the screen – the top screen line and part of the second. The fun part comes by playing. Try different groups of characters: "ioioioioi"..etc produces the effect of three parallel ladders; repeating the asterisk and english pound characters displays a repeating checkerboard effect; "cxcxcxcxcxcxcxc" is pretty interesting, too (all of these were found by experimenting). Type in your name to see what it "looks like". As usual, we leave it to you to find an application for the above bit of foolishness.

Try This

```
10 geta$:if a$="" then printb$;: goto 10
20 b$=b$+a$: printb$;: goto 10
```

Press a few keys, then try some cursor controls. It will eventually die with a ?STRING TOO LONG, but by then you'll be tired of it anyway.

Error–Driven Catalog Routine for VIC/64

This machine–language program sits in the cassette buffer and displays a directory of drive zero whenever a ">" (greater–than) is entered. It works by trapping the syntax–error vector, so it won't bother anyone when it's not in use.

```
LB  10 rem save"0:errcat 64.bas",8
MM  100 rem ** rte/85 - error vector driven catalog
        routine for c64 and vic 20
NJ  110 rem ** press > then (return) for a catalog of
        drive zero
HG  120 for j=828 to 951: read x: poke j,x: next
DD  130 sys(828)
KK  140 rem
PE  150 data 169,  71, 141,   0,   3, 169,   3, 141
JN  160 data   1,   3,  96, 201,  49, 208, 104, 169
GK  170 data   2, 162, 182, 160,   3,  32, 189, 255
FC  180 data 169,   2, 162,   8, 160,   0,  32, 186
BD  190 data 255,  32, 192, 255, 162,   2,  32, 198
ID  200 data 255, 169,  13,  32, 210, 255,  32, 207
GB  210 data 255,  32, 207, 255, 160,   2,  32, 207
CB  220 data 255,  32, 207, 255,  32, 207, 255, 170
DD  230 data  32, 207, 255, 132, 251,  32, 205, 189
IH  235 rem   ok,  ok,  ok,  ok,  ok,  ok,  ok, 221
    Note: use line 235 to change line 230 for vic 20
```

```
LG | 240 data 164, 251, 169,  32,  32, 210, 255,  32
AC | 250 data 207, 255,  32, 210, 255,  32, 183, 255
LI | 260 data 208,  19, 200, 192,  28, 208, 240,  32
CD | 270 data 225, 255, 240,   9, 169,  13,  32, 210
KN | 280 data 255, 160,   0, 240, 201, 169,   2,  32
PK | 290 data 195, 255,  32, 204, 255, 162, 128,  76
ML | 300 data 139, 227,  36,  48
JJ | 305 rem  58, 196,  ok,  ok
```

Note: use line 305 to change line 300 for vic 20

Notes On REVCNT: The Error Recovery Count Variable – CBM Drives

Your drive can tell you quite simply when it is out of alignment. By writing a value of 193 to location REVCNT (see below), your drive will err out immediately when an alignment error occurs. The code and an explanation follows below:

```
1541/2031LP :  print#15, " m-w " chr$(106)chr$(0)chr$(1)
               chr$(193) : rem loc $006a
2040/4040   :  print#15, " m-w " chr$(252)chr$(67)chr$(1)
               chr$(193) : rem loc $43fc
8050/8250   :  print#15, " m-w " chr$(245)chr$(16)chr$(1)
               chr$(193) : rem loc $10f5
```

The Reasons Behind Choosing The Value 193 (Binary 11000001)

A quick note on the 6502 BIT instruction. When a BIT is performed on a memory location, the NEGATION flag is set from bit 7 of the location, and the OVERFLOW flag is set from bit 6 of the location.

A BIT instruction is performed on REVCNT by DOS for two different reasons. First, after a BIT on REVCNT, a BVS is made that branches past a routine that executes a track offset. Second, after a BIT on REVCNT, a BPL is made that branches past a routine that tosses a BUMP onto the job que. These two reasons explain why Bits 7 and 6 were set (192), but still leaves the last bit, Bit 0, unexplained. Look below for the answer.

Whenever an error occurs when reading or writing to disk, the routine is attempted a set number of times before aborting. Location REVCNT holds the key to the number of attempts. The DOS will AND location REVCNT with #$3F, storing the result in the Y register for a counter of the number of attempts. If you were to AND 192 with $3F, the result would be zero:

```
11000000 (192)
00111111 ($3F)
--------
00000000 after ANDing
```

Therefore, in order to not loop through 255 cycles of attempts (DEY, BNE routine), bit 0 has to be set. This gives a total value of 193 (Bits 7, 6, and 0 set)

Original 1541 tip thanks to the Central Coast Commodore Users Group Newsletter – April 2, 1985.

ML Right Justify — Richard Perrit, South Porcupine, Ont.

In Volume 5, Issue 6 we ran this one-line "right justify" for 80-column computers:

```
fori=1to80:print" ■ ";:forj=1to24:print" ■ ":next:next
```

Richard Perrit of South Porcupine, Ontario has since re-written this special effect in machine language. The program is relocatable and can be installed using the BASIC loader below.

```
CK | 10 rem *** right justify 80 ***
GD | 20 rem *** richard perrit  ***
LO | 30 rem *** august 11/85   ***
MJ | 40 :
JE | 50 rem ad=49152 for c-64
GE | 60 rem ad=634   for pet
JL | 70 rem must have 80 columns
EM | 80 :
IL | 110 ad=634:fori=adtoad+31:readx:ch=ch+x
   |     :pokei,x:next
EM | 120 if ch<>4605 then print " !data error! " : stop
FK | 140 data 169,   0, 162,   1, 160,   1, 169,  19
IO | 150 data  32, 210, 255, 169, 148,  32, 210, 255
CO | 160 data 169, 141,  32, 210, 255, 200, 192,  24
AB | 170 data 144, 241, 232, 224,  80, 144, 229,  96
```

Slipped Disks: Speeding up your disk drive — Scott Maclean, Georgetown, Ont.

This article deals with speeding up dual drives – examples are given for the 4040, 8050 and 8250. Unfortunately, the method given here will not work on the 1541, because the method we are using does not exist on the 1541.

In the dual drive memory map, at location $1000 (4096 decimal), to location $1003 (4099 decimal) are 3 interesting variables. (Note: 8250 values also apply to the 8050 drives)

Location		Contents (4040)		Contents (8250)		Label	Description
Hex	Dec	Hex	Dec	Hex	Dec		
$1000	4096	$0A	10	$03	3	ID	Interrupt Delay
$1001	4097	$0D	13	$0D	13	MAD	Motor acceleration delay
$1002	4098	$30	48	$30	48	MCT	Motor cutoff time

We can change the contents of these locations to change the speeds of the different functions of the disk unit. We can

change the value of the Interrupt Delay, which increases or decreases the overall speed of the drive, including the transfer rate of the drive. Very small delay rates will cause read errors and the drive won't read a thing from disk. The most noticeable thing this value changes is the speed at which a "drive bump" occurs. For instance, set this to 5 on a 4040 and then open a file to disk with the drive door open to cause an error. You will hear a buzzing noise instead of the familiar "WHAPWHAPWHAP" noise a 4040 makes. Also affected is the stepping rate, if you send the head from track 1 to track 35, you will notice a significant increase in stepping speed. A safe value for the 4040 is 9, and for the 8050/8250 is 2.

We can also change the Motor Acceleration Delay rate. When you tell the drive to access the disk, it turns on the drive motor, then waits for a certain amount of time for it to accelerate and stabilize to exactly 300 RPM. We can change this value to change how long the startup delay is. Safe values for all drive types is 2. This value has the most visible effect, as it decreases directory search times, and generally speeds all internal disk access up. Using these two functions, you can read the directory from a 4040 with about 1 second of drive motor time. After setting these two locations and requesting a directory, the 4040 will do a drive bump, move to track 18 and seem to stop instantly. However, it will continue sending directory data until it has finished the directory.

The last location is the Motor Cutoff Time. This is the delay the drive uses after a file is closed, or after data stops flowing. Normally, after you finish using the drive, it will whirr for a few seconds longer, even though it isn't doing anything. By changing the value in this location you can control how long it will continue to spin the disk. If you are used to the length the 4040 spins, and you then start to use an 8250, you will notice that the 8250 seems to take forever to stop spinning. Using all three locations, it is possible to change the entire speed characteristics of the drive. Following is a table showing the safe values for each location, followed by a short program that can be used to change the values easily and quickly.

One last note: I would expect that the same method should operate correctly on the SFD–1001, but don't quote me on that as I have never used one of those units.

Location		Lower Limit		Upper Limit	
Hex	Dec	4040	8050/8250	4040	8050/8250
$1000	4096	$0A 10	$03 3	$F5 250	$F7 252
$1001	4097	$02 2	$02 2	$FE 254	$FE 254
$1002	4098	$02 2	$02 2	$FE 254	$FE 254

Editor's Note: The above Lower Limit values may not work on all drives – experiment. Also, speeding up your drive may make it less reliable; don't trust important data or complex disk functions to a hyped–up machine.

```
10 rem **program to change velocity
20 rem **values of dual drives
30 rem **by scott maclean
40 open 1,8,15:rem **open command channel
50 print chr$(147)
60 input " Interrupt Delay " ;id
70 input " Motor Accel. Delay " ;mad
80 input " Motor Cutoff Time " ;mct
90 print#1, " uj " :rem **reset drive
100 print#1, " m–w " chr$(0)chr$(16)chr$(3)chr$(id)
    chr$(mad)chr$(mct)
110 rem **sets up at locations $1000–$1003
120 close 1
```

```
10 rem **quick program to speed up
20 rem **dual drives
30 open 1,8,15:rem **open command channel
40 print#1, " uj " :rem **reset drive
50 print#1, " m–w " chr$(1)chr$(16)chr$(2)chr$(2)chr$(2)
60 close 1
```

I welcome comments on this method, I may be contacted at:

MFP Enterprises
6 Marilyn Crescent
Georgetown, ON
L7G 1K4
Or by modem at (416) 877–7762.

1541ders **Daniel Bingamon, Batavia, Ohio**

When I attempt to open a relative file with a record length of 58 (ASCII code for colon) I get errors. It appears that the 1541 likes to think of the colon as a delimiter and since between the comma and the colon is nothing, you get an error for opening a file of record length zero. Maybe this will give Commodore the hint to tear into their source and fix this along with a few other problems (like SAVE@), if we find enough bugs.

The " UJ" command sent via the command channel is being used by some widely sold software. Some drives (most of them) require three seconds for the reset, but some software only waits one second or less. this causes the computer to " hang up" when further disk commands are given. This can occur when the programmer writes a routine in BASIC, then compiles and does not compensate for the speed increase in the FOR..NEXT time delay loops.

C–64 BASIC STP **Jack Weaver, Miami, FL**

"STP" stands for "Sequential To Program". This is a BASIC STP for those who don't want to STP the M/L way. Refer to Chris Zamara's STP program in Transactor Vol 5, Issue 6.

This routine will enter any program that has been listed to a SEQ file on disk. It uses the Dynamic Keyboard technique from BASIC.

As a dividend, BASIC STP may be used to append or merge several programs together. The individual program lines must have no duplicate numbers or your final program will be a total mess.

A great idea is to have a series of routines, with specific numbering for each category of routine. Call and merge them together with BASIC STP. Build a program of routines, using BASIC STP to do it.

To use it for appending a program or routine to an existing program, you may LOAD Basic STP and list it to the screen. Then LOAD the program you are using as the "master" program. Bring the cursor up to the top line of BASIC STP, and hit RETURN over all the lines, 63990 through line 63999. Now BASIC STP is appended to the program.

RUN 63990, and enter the file name of the routine or program on SEQ file you wish to append or merge with your "master" program. BASIC STP will do just that.

The last step is to delete BASIC STP lines, and SAVE the new program.

```
KK  63990 poke828,169:poke829,0:poke830,76
FM  63991 poke831,49:poke832,243:close4
PG  63992 input " filename " ;f$:open4,8,4,f$
          :get#4,a$,a$:poke829,1:a$ = " "
MP  63993 print " Sqqq poke812,60:poke813,3 qq "
          :ifa$<> " " then63995
PI  63994 get#4,a$
FI  63995 printa$;:ifa$<>chr$(13)then63994
IO  63996 get#4,a$:a = 0:ifst = 0then
          a = asc(a$ + chr$(0))
NB  63997 print " a$ = chr$( " a " ):goto63993
BK  63998 ifstthenpoke829,0:close4:stop
GO  63999 poke198,3:poke631,13:poke632,13
          :poke633,13:print " s " :end
```

The Lottery Companion

When you run out of birthdates, license numbers and hats to pull numbers from, you might want to use this program the next time you play a lottery. It will pick up to ten sets of six numbers, chosen from a pot of 39 or 49, as you choose.

```
OO  100 rem save " 0:lottery " ,8
KA  105 rem ** an evers co-production 1985 **
MJ  110 dim win%(49,10), out$(10): c$ = chr$(147)
JG  115 print c$ " select option "
DF  120 print " 1) lottario 6/39 "
HD  125 print " 2) lotto   6/49 "
NF  130 input x$: if x$< " 1 " or x$> " 2 " then 130
CF  135 lot = 39: if x$ = " 2 " then lot = 49
IN  140 input " output (3) screen (4) printer " ;dv
          : if dv<3 or dv>4 then 140
IG  145 open 1,(dv)
FA  150 input " required # sets (1-10) = " ;max
          : if max<1 or max>10 then 150
HN  155 print#1, " your 6/ " mid$(str$(lot),2) " numbers
          are: " : print#1: print#1
    160 rem assign the random values to the array
KI  165 for try = 1 to max: for pik = 1 to 6
CL  170 v% = rnd(0)*lot + 1: if win%(v%,try) then 170
          : rem loop till un-used #
DH  175 win%(v%,try) = 1: rem flag as used
MA  180 next pik, try
EB  185 rem ** got the numbers - build the strings **
AM  190 for pik = 1 to max: for asn = 1 to lot
MI  195 if win%(asn,pik) then out$(pik) = out$(pik)
          + right$( " [3 spaces] " + str$(asn),4)
AO  200 next asn, pik
HK  205 rem ** all ready - time to print **
DH  210 for spt = 1 to 24 step 4: for prt = 1 to max
EP  215 print#1,mid$(out$(prt),spt,4);
HD  220 next prt: print#1: next spt
MD  225 print#1: close 1: end
```

Gaussian Elimination Routine — **Audrys Vilkas, Goleta, CA**

The following routine is capable of solving up to nine equations in nine unknowns of the form Ax = b. It can also solve or yield information about non-square arrays. It is done entirely off-screen but the user should be aware that a little gentleness in key input is appropriate. The routine occupies 700 or so odd bytes in the raw and is an excellent tutorial for those who study matrix theory.

```
EL  100 rem * gaussian elimination routine *
FI  110 print:input " Row Dimension " ;n:input " Column Dimension " ;m
LO  120 dim a%(n,m + 1),b(n + m + 1): fori = 1ton: forj = 1tom + 1: k = i + j
MN  130 print " a " i;j;: input "  = " ;a(i,j,b(k))
AP  140 print " Q " ;:next:next:print:print " r Next Row Dim " ;n-1; " Next
        Col Dim " ;m-1
NG  150 fori = 1ton:forj = 1tom + 1:printa(i,j,b(k));:next:print:next:print
PP  160 fori = 1ton:forj = 1tom + 1:def fna(i) = -a(i-1,1,b(k))*a(i,j,b(k))
KK  170 def fnb(i) = a(i,1,b(k))*a(i-1,j,b(k)):r = fna(i) + fnb(i)
KA  180 r1 = -a(i-1,1,b(k))*a(i,j,b(k)):r2 = a(i,1,b(k))*a(i-1,j,b(k)):ra = r1 + r2
CD  190 r1 = a(i,1,b(k))*a(i-1,j,b(k)):r2 = a(i,1,b(k))*a(i-1,j,b(k))
        :rb = r1 + r2:r = ra*rb
GN  200 r = fna(i + 1) + fnb(i + 1):printr;:next:print:next:ifm = 1and
        n = 1then220
PP  210 clr:goto110
PH  220 y = a(n,m + 1,b(k))/a(n,m,b(k)):print y; " is a solution " : clr: goto 110
```

The Evil Swords Of Doom!

Beware as the evil sword slices through the screen and wipes any characters unfortunate enough to be in its way. Look out! Here comes another - you never know where the next one will strike. Before long, all characters have been slain by the EVIL SWORDS OF DOOM! Stay tuned until next issue for the conclusion of this exciting tale. (PHHH Gimme a break Chris - KH)

```
10 rem evil swords of doom
20 a$ = " MqMqMqMqMQQQQ | | | | |   q "
30 b$ = "   q  q  q  q   "
40 print chr$(142)
50 print chr$(19)tab(rnd(1)*41)
60 fori = 1to19:printa$;
70 rem delay here if desired
80 next i: print b$;: goto 50
```

Letters

Twinkle Tones: First of all, I would like to say thanks for the great communications issue of Transactor (Vol 6 Iss 02). I enjoyed it very much as I am interested in telecommunications with my C–64.

In your article (Tele–Tone 64) you mentioned the fourth column that belongs to the set of Touch Tones. In the industry Touch Tone is referred by the technical term of 4x4 signalling as the full pad is 4x4. The extra column is used by the US Military in their private Autovon system.

The tones are used in times of emergency to get important phone calls through even though all circuits are busy. The call is given a priority by the fourth column digit. It can override a lower priority call in order to get through. The highest priority call is 'Flash Over' (FO). Then 'Flash' (F) is second. 'Immediate' (I) is next. The fourth priority is 'Priority' (P) and then at the bottom is a call with no priority. However, all of these tones are of absolutely no use to the average caller for those of you with experimentation ideas. Your local Bell exchange will totally ignore them.

As I work with Illinois Bell Telephone, I was interested in Tony Valeri's article. The fact that he included the 'No Such Number' was interesting as I don't know of any company or exchange in the Northern Illinois area that still uses that tone.

One of the things that I have learned in playing with my C–64 is the amount of mis–information available! The amount and type errors in reference material is great and CBM puts out more than their share of it also. The biggest error that comes to mind is the RS–232 tables that are built–in the Kernal. It seems that if you want to run a 1200 bps modem on the C–64, it won't work (until you find out why).

From most reference material available including the Programmer's Reference Guide, you would open the RS–232 channel with the following syntax:

```
OPEN 2,2,0,CHR$(8) + CHR$(0)
```

For one stop bit, 8 bit words, 1200 bps, no parity, and full duplex.

But guess what?? That won't work! It seems that the baud rate table is wrong. Also, the PRG supports two more errors:

a) It infers that the User Defined Baud Rate is not implemented.
b) The formula that they give to figure the User Baud Rate is wrong.

Now I don't pretend to be smart enough to have figured it all out myself. I had deduced enough from what I had read and done with my C–64, that a User Defined Baud Rate was used to make a terminal program run 1200 bps on the C–64. Joe O'Hara at Microtechnic Solutions was kind enough to tell me the 'secret'. It turns out that his associate Rick Sterling had done that math from the ground up and came up with the right figures to make the C–64 work at 1200 bps. The command is:

```
OPEN 2,2,0,CHR$(0) + CHR$(0) + CHR$(57) + CHR$(1)
```

The two CHR$(0)'s activate the User Baud Rate determined by the second two CHR$'s. As shown it's 57 + 1×256 for a total of 313. I'm also told that CHR$(59) may work better if CHR$(57) causes any trouble (ie. 315 total).

I have since been able to modify and use several excellent public domain terminal programs at 1200 bps. And I enjoy it very much as I now can do as much as I used to do at 300 bps in less time! But it seems that my wife does not think I spend any less time with my C–64 though.

Hope that the information is of some use to you and keep up the good work of giving us good Commodore information and programs!.

Lyle R. Giese, Woodstock, Illinois

Thanks for supplying the final piece to the Touch Tone puzzle. It may interest you to know that I thought the extra row was for military use, but without proof and working details, we couldn't risk printing it. With your help, the story is now complete.

Our compliments to Rick Sterling for his detective work with the RS–232 tables. The Inner Space Anthology also suggests the User Rate is unimplemented. Is it possible Mr. Sterling might share his findings?

Disk Risk?: I have read a small amount of advertising information about the SFD 1001 disk drive as a data storage drive to hold your data, but almost nothing is ever said whether or not it can run commercial programs!, excepting one company (Protecto). They claim that if the program is back–upable, then the drive will run the backed–up program. What I need if that is true, it seems that the SFD 1001 would act as if it were a 1541. Am I right in my assumption?

Jim McCoy, Opa–Locka, Florida

The SFD 1001 could be compared to 1/2 of a Commodore 8250 drive. It has the capacity to store up to 1 megabyte of information on a single diskette, and it also has an IEEE port on the back. It cannot act like a 1541 drive, no matter how hard you try. Everything is different. The 1541/2031/2040/4040 type of drives have a limit of 35 tracks on a diskette, with up to 21 sectors per track. The SFD 1001/8050/8250 have up to 154 tracks and a maximum of 29 sectors per track. In simple terms, the drives are not at all compatible.

If Protecto states that if a diskette is back–upable, then the SFD 1001 will run the backed up program, then they are generalizing too much. If you can copy the program over to the SFD 1001 format, then you may be partially there. If the program does not use direct access techniques with the drive, does not have the 1541's ROM and RAM in mind while protecting itself, and does not consider the 1541's disk format for operation, then you may be in luck. Programmers are a unique lot when it comes to squeezing every ounce of storage space out of the 1541. If the drive turns out to be other than a 1541, you may be in trouble. The program will try its magic on the wrong drive and, pronto, system bomb. A rotten end for a program just trying to be nice.

In summation, the SFD 1001 is a good drive. It offers great scads of storage space for anyone who is interested. It would be perfect for bulletin board systems, wordprocessor users, databases that only use the common file types like SEQ, REL, etc., also games, utilities, and programs that don't use the disk for any tasks, and just great for anyone who requires more storage room than the 1541 offers, plus an increase in operating speed. But a price has to be paid. The SFD 1001 is an IEEE drive, therefore you need an IEEE interface for your 64 to use it. IEEE links usually live in the 64's cartridge port, therefore, you lose the port. And some interfere with the 64's architecture which may confuse some programs, usually those that demand total control of the machine and thus contend for memory that the interface occupies. In general, though, most software people and hardware people (ie link designers) have addressed this problem and most of the more well developed packages can cope with these configurations.

Following that, however, the format is not at all similar to the 1541. The SFD cannot read 1541 disks, and unless you have some way to read from the 'serial' 1541 and transfer to the 'parallel' SFD, then you may have some trouble just getting your programs onto SFD formatted disks, short of using a 4040 to do the transfer (the 4040 is 1541 and IEEE compatible and could be used to make an easy transfer). The only drive that is compatible with the SFD is the Commodore 8250, which went out of production a short while ago. The next "almost" compatible drive is the 8050 with DOS 2.7 ROMs. The trouble with this is the SFD 1001 has two heads per diskette, or double–sided, while the 8050 is only single sided. The SFD 1001 gives a total capacity of 4133 blocks per diskette. The 8050, 2052 blocks. Therefore, if you write past the half way point onto the "other" side, you cannot expect to read it with the 8050. If you don't require "diskette protability" then you need not be concerned here though.

This may sound like a mouthfull, but on the other hand should be second nature when considering such purchases. Get to know the buss types and format differences from one piece of equipment to the next. With often just a few facts one can rehearse the events following a major system modification and usually determine its success without actually making any changes.

Microfiche Interest: I have just finished reading the September issue of The Transactor. This is the third issue I have bought and, although I haven't written to any other magazine, I feel compelled to write to yours.

Although several other magazines are published that concern themselves with the Commodore computer (usually intensifying themselves on the C64), they can't always be taken seriously as they tend to appeal to too broad an interest and ability level. This is done honorably enough by the publishers to try to help everyone who buys their publication, however it tends to penalize those of us who have gone beyond the basics and to some degree the intermediary level of computing. That is why I am so fascinated by your magazine, it appeals to the higher level user who needs more in depth knowledge without all the in–depth explanation. That much said, I would like to encourage you not to sacrifice the quality and integrity of your magazine in an effort to do what everyone else is trying to do, appeal to everyone. It's like we were all told as kids, "If everyone jumped off a bridge, would you?"

This letter was actually written in response to a note in 'News BRK' about the possibility of The Transactor appearing in microfiche. I would like very much to see this happen as it might make it a little easier to get a copy of your magazine in light of the fact that it can be extremely hard to find at this time. I do intend to subscribe myself, but I know that a greater availability of a good publication benefits all Commodore computer owners. I suppose this brings up the never ending problems of copyright violations and, while I sympathize with that view, I think that it's terrible that we have to sacrifice the education of a large population of users in fear of the few among us who insist on trying to get a free ride. But I see an even more important advantage of this happening. There are countless articles that were published in the early days of The Transactor's existence that are unavailable to myself and any other people who are newly acquainted with your magazine. If these old issues become available by microfiche, it would provide a great service to those of us who crave all the knowledge we can find about our 64's and other Commodore computers to which we are fiercely devotes.

I would like to briefly summarize the rest of the things that I enjoy about your magazine. First, the emphasis on machine language. This kind of information and free use of such a vital and important part of the computer is sadly lacking in other popular publications. Secondly, the more in–depth look at the 1541, truly a mysterious drive which has so little available documentation that it becomes frustrating to use when something goes wrong, a minor problem becomes a major catastrophe. And third, your policy on published programs which is far better than the competition. If I type in a program and find it useful, I am glad that I am free to give a copy to a friend to make his life easier. It's this kind of exchange that unites users and forms a more close knit bunch of enthusiasts that accomplish things together to make computing easier and more accessible to all of us.

In closing, please retain your high standards in the publication of your magazine, but don't get so big that you're forced to conform to the lower standards of your competitors. Keep up the good work! Tim Blazier, Elgin, Illinois

Sorry, but effective next issue we will start to cover a whole new horizon with our magazine, Basic Backet Weaving. In our Basket Weaving issue, renamed The Transbasket, we will address such problems as: The advantages of weaving to the left instead of the right. Our thoughts on putting a plastic bag at the bottom of the basket. Instructions on how to put a plant in your finished basket. And finally, cheap gifts for Christmas, Baskets!!. Talk about fun. Can't you wait??

I couldn't resist. But The Transactor will continue to be produced at the level we have come to expect of ourselves – if we were not learning from our own work and research, we would get bored and probably change jobs. We like it this way, and letters like yours plus conversations with our readers is a large part of our motivation.

Microfiche: Your letter is the very first that we have received mentioning it. Perhaps very few people read our News BRK section. Or perhaps people would like to see The Transactor on microfiche, but don't write in feeling there wouldn't be enough others. Whatever the story, we want to go on microfiche, but can't justify it until a demand is clearly shown. If any of you are really interested, as Tim is, then drop us a letter. Once we are sure, then microfiche will be on its way.

An organized submission: Hello from sunny Vancouver! I realize that this program (Yellow Pages Directory Organizer) is out of sync with your editorial schedule, but I am rather proud of it, and it is public domain. I would be most pleased if your would be most pleased if you would include it on your monthly disk, should you find the room. In any case, you should find it useful yourselves.

Complete documentation is included in the program.

I am a technician with Memorex. I work with 800 meg hard drives, 200 inch/sec tape drives, etc. I've been involved with Commodore computers since about 8 years ago when I bought a second-hand PET 2001, with the old ROMs no less! A friend I work with, Larry Philips, said he spoke with you folks at Marca and you would like some authors. I'm busy sharpening my pencil (figuratively, of course) and hope to submit an article soon.

I enjoy your magazine very much, and wait impatiently for it to arrive on my local dealer's shelf each month. I'd buy a subscription, but Canada Post is notoriously slow, especially, it seems, with any magazine I subscribe to.

Yours for more public domain software.
Rick Morris, Burnaby, B.C.

Thanks to Rick, the Yellow Pages program will be appearing on all future Transactor diskettes. The features of this program are pretty impressive, such as; allows the C64–1541 user to move filenames about easily within the directory. For an encore, files can be selectively scratch protected and un–scratch protected (locked and unlocked). Plus, you can scratch and un–scratch files at will. If you want, you can also put a bar separator between filenames that gives the directory listing a more pleasant and logical appearance. But you have to wait for his program to make the rounds before finding out everything this beauty does.

Thanks for all, Rick, and thank you Larry for the kind reference - much obliged.

Profile cosmetic surgery: I enjoyed reading Dr. John Ross's article regarding " Speeding Up Your BASIC Programs " , which appeared in the recent issue (Volume 6 Issue 3). Another fine article in a fine publication.

After adapting his program for the Vic and putting it to use, it is indeed a valuable utility.

As the author pointed out, it was written for a CBM 8032 but should be readily adaptable to other CBM models. I wonder how many of your readers who are Vic and 64 users, who may not yet be able to do an adaptation, will miss out on this valuable utility.

With this thought I sent along the Vic and 64 adaptation. The Vic user of course requires at least 3k expansion, since the Dr.'s program takes 4k for storage. The listing (for program 3) is for the Vic; the 64 user will require changing the (bold face) 191's to 49's . . . The only change required for the program 2 is the variable LO in line 130; change to:

```
lo = peek(56)*256
```

The final change, from the original article, is that of the three SYS; they are SYS 828, SYS 852, SYS 865 respectively.

If you are using a Commodore printer, you may want to change chr$(223) in program 2, to chr$(166) to simulate a better graph.
R.C. Marcus, Agincourt, Ontario

```
CI | 100 rem profiler loader - for vic/64
NN | 110 poke 55,0: poke 56,peek(56)-16: clr
OB | 120 read n,l: for i=1 to n: read x: poke l,x: l=l+1
   |     : next: end
OB | 130 data  87, 828
FD | 140 data 165,  56, 133,   1, 169,   0, 133,   0
BJ | 150 data 168, 145,   0, 230,   0, 208, 250, 230
PI | 160 data   1, 166,   1, 224, 128, 208, 242,  96
CI | 170 data 120, 169, 110, 141,  20,   3, 169,   3
OP | 180 data 141,  21,   3,  88,  96, 120, 169, 191
LL | 190 data 141,  20,   3, 169, 234, 141,  21,   3
NJ | 200 data  88,  96, 160,   0, 165,  57, 133,   0
MB | 210 data 165,  58,   5,  56, 133,   1, 177,   0
NE | 220 data 170, 232, 138, 145,   0, 208,  13, 165
OO | 230 data   1,   9,   8, 133,   1, 177,   0, 170
OG | 240 data 232, 138, 145,   0,  76, 191, 234
```

You're right. Dr. Ross did a smash up job with his article and program. And it did deserve a Vic–20/C64 rewrite. Thanks for the adaptation. It's appreciated.

Case of the missing Space: Several members of our User's Group and myself have typed in the programs of the above listed article of the Sept. 1985 issue of The Transactor, and have encountered some problems with the programs. In the first program on the alignment check we kept getting "reading track 35 – error 70 – no channel", "drive has failed alignment check", regardless of the drive we tested. On running a trace of the program, we found that lines #135 and #155 don't seem to be executing. The program runs good from 100 to 130, going to gosub 160 and continuing to 185, where it ends.

We would appreciate knowing if there is a printing error or possible an explanation of lines 110, 125, and 135. We also experienced problems with line 30 of the second short program and would appreciate an explanation of line 30.

William Nowak, Mohawk Valley Commodore Users Group
Tribes Hill, New York

The problems that you are having are perhaps partially due to the programs as listed. Mr. Clutter used a syntax of Block–Read that is seldom used, but works fine all the same. The statement:

```
print#15," b–r 2 0 1 9"
```

could also have been written:

```
print#15," b–r "2;0;1;9
```

Delimiters of spaces when inside quotes (as Ed Clutter used), or semicolons when outside quotes are up to the users discretion. Either method sends the same information to the DOS as long as the individual parameters can be distinguished by the DOS. Perhaps we should be a little more careful as the typesetter doesn't show spaces as clearly as necessary under such circumstances. Below is a listing of the original program, followed by the lines which you may want to alter to ensure proper delimiting.

1541 Alignment

```
100 d=8: rem d = device number
105 open 15,d,15: open 2,d,2," #"
110 print#15," m–w" chr$(0)chr$(0)chr$(1)chr$(192)
115 t=35: h$=" –"
120 t$=str$(t)
125 print#15," b–r 2 0" t "9"
130 gosub 160
135 print#15," b–r 2 0 1 9"
140 t$=str$(1)
145 gosub 160
150 t=t–1: if t>0 then 120
155 close2: close15: end
160 print: print " reading track " h$;t$,
165 input#15,a$,b$,c$,d$
170 print a$;h$;b$;h$;c$;h$;d$
175 if val(a$)<2 then return
180 print " drive has failed alignment check"
185 goto 155
```

Track 00 Adjustment - Move Head To Track 1, Sector 0

```
10 open 15,8,15
20 open 2,8,2," #"
30 print#15," b–r 2 0 1 0"
```

Make the following changes for the alternate syntax. Notice that fundamentally these lines are no different except for the delimiting of parameters.

```
125 print#15," b–r "2;0;t;9
135 print#15," b–r "2;0;1;9

30 print#15," b–r "2;0;1;0
```

By the way, in line 110,

```
print#15," m–w" chr$(0)chr$(0)chr$(1)chr$(192)
```

was used to put a Bump on the 1541's job que to Bump the head into position on Track 1, Sector 0. That is what the clattering noise is all about when the program first fires up. You could delete this line if you want, and the program would still work Ok.

Disk Woes: I have a few things on the agenda in this letter, any of which you may publish in future editions of your magazine.

I Volume 6, Issue 03, John Brunner of Chicago, Illinois, makes some suggestions about your advertising. I agree whole–heartedly with him. Word–of–mouth is the best kind of advertising anyone can get! I also enjoy reading ads, as they give me an idea of what's out there on the market. I like the idea of keeping all advertising in one section of a magazine, almost like a catalogue. In fact, I like your magazine so much, that I've posted an ad on my Bulletin Board System, here in Ottawa, Ontario, as well as sending in my first renewal fee to The Transactor. (Besides, I could use some advertising, too!)

I have several 1541's and two SFD–1001 (1 Mbyte) drives, and have run into a particular difficult problem, with which I hope someone may be able to help me. One of my important disks was accidently NEW'd but I caught it during the head–rattling and the disk came out with only one block destroyed . . . TRK 18, SEC 1! All programs where the filename is on any other directory block load and run normally. Unfortunately, I don't know how to find the first block of the programs where the filenames reside on the damaged block, except the first file. (DOS always puts first file to start on TRK 17, SEC 0) I'm assuming that all the programs are still intact. Is there something out there that can help me piece this directory block back together again?

Chris K. Weisner
Ottawa Mail Forwarding Services
Box 793 Station 'B'
Ottawa, Ontario, K1P 5P8
BBS (613) 830–2923

It was awfully nice of you to advertise The Transactor on your BBS. We LOVE free advertising. Also, let's hope that our printing your complete name, address, and BBS # after helps you too.

Now, about your partially new'd diskette. (nude?) If the head is caught mid–stream during a full new (during the rattling phase), then little damage should have occurred to the diskette. Even if the rattling had just stopped, and you popped the diskette out of the drive, little damage again. During a full new (with disk name and new id), the drive starts at Track 1, Sector 0, and works its way up sequentially to Track 35, Sector 16. You could not lose Track 18, Sector 1 that fast.

If you performed a quick new (no id), and left it for its duration, then you would have retained every directory block but Track 18, Sector 1, and the BAM would have been re–written. As you know, after a quick new the diskette appears clean. But after changing two bytes (the link pointers to the next directory block on the first directory block), you could find all but the first eight filenames back. From this point, a validation would bring back those files. Then a comparison for allocated sectors versus sectors with data but not allocated begins. You could rebuild, using false filenames, in this way. But quick NEW's are silent – they do not rattle the head about. Something weird must have happened during your full new.

If you turned the drive off during the head rattling stage, then turned the power back on with the diskette still in the drive, there is a pretty good chance that the power surge and your head racing across the disk surface caused the glitch, not your ill fated new. Your only recourse now is to salvage all you can from the diskette by rebuilding it with whatever disk doctor type program you can find. Several exist, and some have fairly automatic features for doing just what you need – some are even in the public domain. You might also obtain some information on disk format which could help when deciphering data (Anthology page 47 to 49) Good luck.

Remotely Noteworthy: First, I would like to thank all of you for one of the best Commodore computer related magazines there is. I have gotten more information from my past year's subscription to you than from any other magazine. In your networking and communications issue, there was a program called Remote–64 which allowed use of the computer through the C–64 RS–232 port. I knew I could use this with my BBS, and now after typing in the source, modifying the code, etc., I have started to learn assembly language. I now have the routine included in by BBS so I can call remote to do updating. I just wanted to say thanks for the good work and please keep it up!!! Your address/subscription info has been put on by BBS for all users who want to make a good investment.

Joe Minuni, Royersford, Pennsylvania

When Chris was writing Remote–64, he told me that it would be perfect for a bulletin board system. Since then he has been waiting for the news you have given. Thanks for making Chris's day.

The Error Of Our Ways: More Often Oops Than Bloops

As our sales figures continue to climb, so do the amount of letters we receive. We appreciate these letters, because it keeps us informed of your likes and disklikes regarding our magazine. Unfortunately, the number of complaints regarding our program listings have also increased. At times, the complaints are valid. On occasion, errors have been known to appear somewhere between the time we edit the articles, and when all is ready for print. A messed up byte over the phone lines, an error due to an incorrect translation, or some place other than we are looking. In truth, errors do slip by, but all too often they are so insignificant that we can't even justify mentioning them in a later issue.

Now, about 90 percent of all letters received can have their errors traced back to keying the programs in. Complete program lines missing, periods instead of commas between elements in a data statement, mispelled variable names, and a multitude of other equally avoidable errors. Recently, I received a letter that really let us have it for three mistakes. The first was ours, but it was only a missing quotation mark after a print statement. An easy mistake to correct, considering that the program was in our Bits and Pieces column.

The other two belonged to our reader. But the real rub came when the reader stated emphatically that we were in the wrong. A difficult form of criticism to swallow. I won't detail the errors here for the same reason we find it difficult to answer such letters publicly.

Instead of belabouring the point with similar stories, all I ask is for you to check your work more closely before coming to the conclusion that we messed up. We all make typing errors, and depending on the hour or several other affecting conditions, an error can be staring you in the face and you wouldn't see it if it punched you in the nose. Believe me, I know! Sometimes I key an entire line over, or have someone else take a look, or even just explaining it to someone else can make the mistake pop off the page. Although not perfect, we do scrutinize as best we can. Every program is tested within reason and the listings go straight into the typesetter much like you send one to your printer. So have one more look – it will take you less time than writing.

Thanks for your time, and please keep those letters coming.

Richard Evers, Editor

TransBASIC Installment #6

Nick Sullivan
Scarborough, Ont.

This instalment of TransBASIC presents a grab–bag of new modules, some of which contain general purpose routines that could be used again in the future. The first module we'll look at is a very long one, USE, which appears as Program 1.

USE is a fast merge command that will merge modules (or BASIC programs) much more quickly than the ADD command we have been using so far. In addition, it will automatically update line 95 of the TransBASIC kernel (which gives the number of statements and functions in the dialect) using the information contained in line 2 of the merged module. The presence of this feature, which proved unexpectedly hard to code, is largely responsible for making Program 1 so long.

There are two differences between the merge algorithm in USE and the standard one found in routines like ADD. Most merges read in a program line from disk, and merge it individually, using the same routines BASIC uses when you enter a new line from the keyboard. This is a four step procedure: 1) search the program in memory for a line with the same number and delete it if found (moving all higher–numbered lines downward to close the gap); 2) rechain the program, and perform CLR; 3) open up a space to accommodate the new line (moving all higher numbered lines upward again), and insert the new line; 4) again rechain the program, and perform CLR.

Anyone who has added a line to a long program knows that the above procedure is by no means instantaneous, but can take a good second or two before the cursor returns. When an entire program, subroutine or TransBASIC module has to be merged in this way, you can be left drumming your fingers on the desk for quite a while before the work is finally done. USE sacrifices the convenience of the ROM routines in favour of an approach designed specifically for merging rather than entering a single line at the keyboard. It goes like this: Lines are read into a special buffer in free memory, and their numbers are compared with a line number of the program in memory (the main program). If the new line number is less than the line number in the main program, another line is read and added to the buffer, and the process repeats. If the line numbers are equal, meaning that the line in the main program will be deleted, the pointer into the main program is advanced to the line following, a new line is read from disk, and the process repeats. If the new line number is greater than the one in memory, the higher–numbered lines of the main program are moved up or down the required number of bytes, the buffer is copied into the main program, and the process repeats. The rechain and CLR step is performed only at the end of the merge. The gain in efficiency from this method results in merges that are virtually as fast as regular program loads.

The first thing you should do with the module is replace the "TB/ADD.OBJ" file used for constructing TransBASIC dialects with a new file called "TB/USE.OBJ". To do this, use the following procedure:

1) Load and run the program "TRANSBASIC", which sets up "TB/ADD.OBJ" and loads the TransBASIC kernel.
2) Merge the USE module with the command: ADD "USE
3) Alter line 95 to: 95 XTRA .BYTE 3,0
4) Assemble the source file with PAL or similar.
5) Save the resulting object file as "TB/USE.OBJ"
6) Load the "TRANSBASIC" program again, alter line 130 to: 130 A = 1: LOAD "TB/USE.OBJ",8,1 , and resave it.

Three of the subroutines in the USE module may find use elsewhere. One is the memory block move routine MVMEM (lines 8250 to 8414). To use this routine, set up the pointers MVSTRT, MVEND and MVDEST with the appropriate addresses for the area you wish to move. The instruction SYS MVM2 will perform the move. Sometimes it is convenient to make MVEND point to the first free byte beyond the move area, rather than the last byte within it. If you do this, call the move routine with JSR MVMEM, which subtracts one from MVEND then falls through into the main routine. If MVSTRT is greater than or equal to MVEND, or if MVSTRT is equal to MVDEST, no move will be performed, but no error is generated.

The second subroutine that might prove useful is DELINS (lines 8054–8172), which in turn makes use of MVMEM. DELINS deletes text between the addresses pointed to by SDPTR and T3/T4, and replaces it with text between SIPTR and T5/T6. The start–of–variables pointer at $2D/2E is taken to mark the first free byte beyond the affected memory area. In the case of a BASIC program, this is what you would want. If you use DELINS to modify some other part of memory, you would save the start–of–variables pointer, write the appropriate address into $2D/2E, call DELINS, then reload the start–of–variables pointer before returning to BASIC. By the way, any time you change a BASIC program from machine language, you should always rechain and perform CLR before you return control. JSR $A659: JSR $A533 will do this for you. Alternatively, if you are returning to direct mode, you can do the whole operation by exiting with: JMP $A52A.

The third subroutine is a very short one called ERRPGM. Its purpose is to generate a ?SYNTAX ERROR if it is called in program mode rather than direct mode. The main use for this is in commands like USE that alter the program in memory. There is no setup required, just JSR ERRPGM.

The module MOVE & FILL (Program 2) provides two commands that are more commonly found in machine language monitors: move a block of memory, and fill memory with a value. The FILL command uses a subroutine called MEMFIL that you might want to use if you're writing a command to zero

an array or a high-res screen, for example to fill in a portion of colour memory or screen memory. There are several entry points, depending on the way you want to set up your parameters. The start address of the area to be filled can be supplied either in .Y/.A (JSR MEMFIL or JSR MEMF1) or in T3/T4 (JSR MEMF2 or JSR MEMF3). The size of the area to be filled can be specified as either the end address of the area (JSR MEMF1 or JSR MEMF2) or the number of bytes to be filled (JSR MEMFIL or JSR MEMF3). The routine will get this parameter from $14/15. In all cases, the value with which memory is to be filled is supplied in the X register. The MEMFIL routine will exit without doing anything if the start address is greater than the end address, but will return an ?ILLEGAL QUANTITY ERROR if you ask it to fill more bytes than are available between the start address and $FFFF.

The third module this month is DOS SUPPORT (Program 3), by Darren Spruyt of Gravenhurst, Ontario, which supplies TransBASIC with a battery of commands similar to those in the DOS WEDGE. Darren has written these commands in an interesting way that avoids the need to open a file in the 64. You might find it useful to study his coding to see how this trick is done.

One problem with very large BASIC programs is the time taken by the interpreter to locate destination addresses for GOTOs and GOSUBs, which can severely impair performance, especially in the case of subroutine calls from within loops. This problem is addressed in the LINE CALC module (Program 4), which allows you to save time by calculating jump addresses in advance. The LINE(function in this module returns the address of a specified program line. You might use this in a number of ways, but for the purposes of this module you are expected to assign it to an integer variable. The statements JUMP (equivalent to GOTO) and CALL (equivalent to GOSUB) make use of the address stored in the variable to go directly to the line without having to search through the program to find it.

Program 5, the BEEP module, provides a convenient way of generating a beep tone of a pitch and duration specified in the command parameters (the default is a very short C in octave 5). You can use this to give audible feedback for key presses, for example, or even to generate simple sound effects, as in the following little routine:

```
100 FOR I=1 TO 50
110 BEEP 7,RND(1)*2400+2400
120 NEXT
```

BEEP uses voice 3 of the SID chip; the other voices are not affected, except that the volume is set to 15 and the filters are turned off.

The last program this time is not a TransBASIC module, but a little BASIC/ML routine for those who use Brad Templeton's POWER and PAL (from ProLine Software) in their program development. The program is called STRIPPER (Program 6), and its purpose is to remove comments from a PAL source program in memory (a long job if you do it by hand). As shown, you invoke the machine language with SYS 900, but the code is relocatable if you want to put it somewhere else. Keep in mind that it only works if you have POWER/MOREPOWER in memory.

New Commands

This part of the TransBASIC column is devoted to describing the new commands that will be added each issue. The descriptions follow a standard format:

The first line gives the command keyword, the type (statement or function), and a three digit serial number.

The second line gives the line range allotted to the execution routine for the command.

The third line gives the module in which the command is included.

The fourth line (and the following lines, if necessary) demonstrate the command syntax.

The remaining lines describe the command.

USE (Type: Statement Cat #: 117)
Line Range: 7192-8052
Module: USE
Example: USE "MOVE & FILL
Example: USE "CURSOR POSITION",9
Like the ADD statement introduced in instalment #1, this command merges a program in memory with one from disk. A device number may be specified, as in the second example; otherwise, the device set by the DEVICE statement (qv) is used, with a default of 8. If the program being merged is a TransBASIC module, with a line 2 in the standard format giving the number of statements and functions; and if the program in memory has the TransBASIC kernal line 95, labelled XTRA, giving the total number of statements and functions; then the USE command will automatically update line 95 using the data in the new line 2. USE is illegal in program mode, generating a ?SYNTAX ERROR.

MOVE (Type: Statement Cat #: 118)
Line Range: 8174-8248
Module: MOVE & FILL
Example: MOVE 1024,1523,1524: REM COPY TOP OF SCREEN TO BOTTOM
Example: MOVE 53248;4096,12288: REM COPY CHARS TO RAM
This is a standard block move, like the .T command of a monitor. The syntax of the first example is comparable to that used by most monitors: the first parameter is the address of the first byte in the block to be moved; the second parameter is the address of the last byte; and the third parameter is the destination address of the move. The parameters are separated by commas. The second example uses an alternative syntax. Here the second parameter is the number of bytes to be moved, and the first separator is a semicolon instead of a comma. If the parameters do not make sense (for example, if the end address is greater than the start address), no move takes place, but an error is not generated. Also, if the destination address is the same as the start address, no move takes place. You therefore cannot use this command to directly copy the BASIC ROM into RAM, for example.

FILL (Type: Statement Cat #: 119)
Line Range: 8504–8558
Module: MOVE & FILL
Example: FILL 832,1023: REM CLEAR CASSETTE BUFFER
Example: FILL 631;10,13: REM PACK KEYBOARD BUFFER WITH RETURNS
This statement fills an area of memory with a specified value. In the first example, the two parameters, separated by a comma, correspond to the start and end addresses of the area to be filled. In the second example, where the separator is a semicolon, the second parameter gives the number of bytes to fill. The third parameter, if present, specifies the value with which memory is to be filled; the default value is 0.

CAT (Type: Statement Cat #: 123)
Line Range: 8644–8740
Module: DOS SUPPORT
Example: CAT
This statement lists a disk directory to the current output device. Programs in memory are not affected.

DOS (Type: Statement Cat #: 124)
Line Range: 8742–8764
Module: DOS SUPPORT
Example: DOS "S0:ITCHFILE"
This statement sends a command to disk.

DEV (Type: Statement Cat #: 125)
Line Range: 8766–8782
Module: DOS SUPPORT
Example: DEV 9
This statement sets the device number for the other disk commands in the DOS SUPPORT module, and for the USE statement (qv). Allowable device numbers are in the range 8–11. The default device number is 8.

DLOAD (Type: Statement Cat #: 126)
Line Range: 8808–8812
Module: DOS SUPPORT
Example: DLOAD "0:MURPHY
This statement loads the named file from disk, using the current device number.

DSAVE (Type: Statement Cat #: 127)
Line Range: 8814–8818
Module: DOS SUPPORT
Example: DSAVE "0:MURPHY.V2
This statement saves the named file to disk, using the current device number.

DS$ (Type: Function Cat #: 128)
Line Range: 8598–8616
Module: DOS SUPPORT
Example: PRINT DS$
This function returns the disk error channel string, and clears the channel (i.e. a subsequent call, with no disk operation intervening, would return "00,OK,00,00").

DS (Type: Function Cat #: 129)
Line Range: 8618–8642
Module: DOS SUPPORT
Example: U = DS
This function returns the error number from the disk error channel, and clears the channel (i.e. a subsequent call, with no disk operation intervening, would return 0). It is equivalent to: VAL(LEFT$(DS$,2))

JUMP (Type: Statement Cat #: 130)
Line Range: 8846–8868
Module: LINE CALC
Example: JUMP QUIT%
The argument of this statement is an integer variable whose value is the address of a BASIC program line. The effect is the same as a GOTO, but is generally faster, considerably so in long programs.

CALL (Type: Statement Cat #: 131)
Line Range: 8870–8900
Module: LINE CALC
Example: GOSUB JSTK%
The argument of this statement is an integer variable whose value is the address of a BASIC program line. The effect is the same as a GOSUB, but is generally faster, considerably so in long programs.

LINE((Type: Statement Cat #: 132)
Line Range: 8902–8964)
Module: LINE CALC
Example: QUIT% = LINE(5000)
Example: QUIT% = LINE(GOTO 5000)
Example: J2STK% = LINE(J1STK% + 100)
This function returns the address of the BASIC line whose line number is returned by the argument expression. This can be a simple line number (example 1), or any other expression (example 3). The keyword GOTO will be ignored if it is used before the line number (example 2). Its purpose is to allow automatic renumbering of the line number with a renumbering utility. If the referenced line number does not exist, the function returns a value of zero.

BEEP (Type: Statement Cat #: 133)
Line Range: 8966–9042
Module: BEEP
Example: BEEP
Example: BEEP 6
Example: BEEP 16,3034
This statement produces a tone from the SID chip, using the sawtooth waveform in voice 3. The volume and sustain are set to 15; the attack, decay and release are set to 0, and filtering is turned off. Without parameters (example 1), the tone produced is a very short beep with a pitch of C in octave 5. The duration of the beep can be set with the first parameter (example 2), which will lengthen the beep by a factor of the parameter value plus 1. The pitch is set with the second parameter (example 3). Thus the default beep is equivalent to: BEEP 0,8583

Program 1: USE

Because of the sheer size of USE, we were rather doubtful anyone would actually type it in. In fact I'm not sure why we even bothered to show the Verifizer codes (force of habit, I suppose). However we did want to include it for the sake of documentation. We apologize for the small type, but what you see is a compromise; the source for USE couldn't possibly have been printed at regular size. -M.Ed.

```
HJ 0 rem use (june 18/85)          :
FH 1 :
AI 2 rem  1 statement, 0 functions
HH 3 :
DO 4 rem keyword characters: 3
JH 5 :
NJ 6 rem keyword     routine     line     ser #
FM 7 rem use         uze         7192     117
MH 8 :
GN 9 rem u/mvmem (8250/120)
GM 10 rem u/delins (8054/122)
BA 11 rem u/errpgm (9150/135)
AI 12 :
LD 13 rem ================================
CI 14 :
AF 39 setlfs   =   $ffba
KD 40 setnam   =   $ffbd
BE 41 open     =   $e1c1
NI 42 chkin    =   $e11e
NI 43 close    =   $e1cc
IB 44 clrchn   =   $ffcc
JK 45 getin    =   $e124
EK 46 ;
MI 132 .asc "usE"
KK 1132 .word uze-1
PC 7192 uze       jsr   errmem    ;check direct mode
IF 7194           lda   #2        ;make space for ,p
BI 7196           jsr   $b4f4
KM 7198           jsr   $ad9e     ;evaluate filename
GH 7200           jsr   $b6a3
MI 7202           jsr   $b4f4     ;make space
MF 7204           tay             ;test null
MB 7206           bne   uz1       ;no
NJ 7208           jmp   $af08     ;'syntax error'
KC 7210 uz1       lda   #","      ;add ,p
DD 7212           sta   ($33),y
OA 7214           iny
HH 7216           lda   #"p"
JD 7218           sta   ($33),y
PJ 7220           iny             ;push filename
LB 7222           tya             ; length
EL 7224           pha
LH 7226           lda   $33       ;push filename addr
IL 7228           pha
MM 7230           lda   $34
ML 7232           pha
HF 7234           jsr   $79       ;test dev parameter
IF 7236           beq   uz4       ;no
ID 7238           jsr   $aefd     ;check for comma
NP 7240           jsr   $b79e     ;evaluate device #
OH 7242           .byte $2c       ;'bit'
PC 7244 uz4       ldx   device    ;default device
IA 7246           stx   t2
DJ 7248           lda   #$62      ;close 98
HM 7250           jsr   close
OL 7252           lda   #0        ;open 98,dv,15
HB 7254           jsr   setnam
FJ 7256           lda   #$62
GN 7258           ldx   t2
HD 7260           ldy   #$0f
IC 7262           jsr   setlfs
EN 7264           jsr   open
JO 7266           lda   t2        ;test dev present
KK 7268           jsr   $ffb1     ; (listen)
MP 7270           jsr   $ffae     ; (unlisten)
LP 7272           lda   $90       ; (check status)
JK 7274           beq   uz5
NL 7276           ldx   #5
JL 7278           jmp   ($300)    ;'dev not present'
EP 7280 uz5       jsr   dskchk    ;check error chan
NL 7282           lda   #$63      ;close 99
JO 7284           jsr   close
NH 7286           pla             ;pull filename data
HE 7288           tay
CA 7290           pla
HE 7292           tax
GA 7294           pla
GB 7296           jsr   setnam    ;open 99,dv,99,
JL 7298           lda   #$63      ; "filename,p"
AA 7300           ldx   t2
FF 7302           tay
CF 7304           jsr   setlfs
OP 7306           jsr   open
HA 7308           jsr   dskchk    ;check error chan
EF 7310           jsr   clrchn    ;discard start addr
NC 7312           ldx   #$63
NI 7314           jsr   chkin
GL 7316           jsr   prgget
IL 7318           jsr   prgget
LL 7320           lda   $2b       ;create ptr into
IM 7322           ldy   $2c       ; program in memory
GL 7324           sta   t3        ; at t3/t4
AG 7326           sty   t4
JD 7328           lda   #0        ;clear flags
LO 7330           sta   uzf1      ;end of disk prg
ON 7332           sta   uzf2      ;lines to delete
NI 7334           sta   uzf3      ;update line 95
GJ 7336           sta   stmctr    ;count new stmts
MG 7338           sta   functr    ;count new funcs
IN 7340 uz6       jsr   makbuf    ;create receive-bfr
FN 7342 uz7       sec             ;test buffer can
DC 7344           lda   $37       ; accommodate new
BK 7346           sbc   t5        ; program line
OE 7348           lda   $38
OO 7350           sbc   t6
HL 7352           sbc   #2
PL 7354           bcs   uz12      ;yes
LA 7356           lda   linflg    ;test lines in bfr
ON 7358           beq   uz8       ;no
PC 7360           jsr   dlz       ;merge lines
DL 7362           jmp   uz6       ;create new buffer
HO 7364 uz8       jsr   uz11      ;close files, clr
CB 7366           jmp   $a435     ;'out of memory'
DM 7368 uz9       lda   linflg    ;test lines in bfr
BI 7370           beq   uz10      ;no
LD 7372           jsr   dlz       ;merge lines
KL 7374 uz10      jsr   chkl95    ;handle line 95
CN 7376           jsr   uz11      ;wrap up
LK 7378           jmp   $a474     ;exit to 'ready'
II 7380 uz11      jsr   clrchn
DB 7382           lda   #$62
NE 7384           jsr   close
MH 7386           jsr   $ffe7     ;'clall'
NH 7388           jsr   $a533     ;rechain
ID 7390           jmp   $a659     ;clr
EH 7392 uz12      jsr   getlin    ;get disk prg line
PN 7394           lda   uzf1      ;test if final line
CO 7396           bne   uz9       ;yes
HC 7398           jsr   chkl2     ;handle line 2
PE 7400 uz13      jsr   compar    ;test dsk line # <
LB 7402           bcc   uz15      ; prg line # - no
FK 7404           php             ;push compare flags
FI 7406           jsr   updabp    ;adv buffer pointer
JK 7408           sta   linflg    ;set bfr-used flag
JD 7410           plp             ;test line #s equal
KM 7412           bne   uz7       ;no, more from disk
KM 7414           lda   uzf2      ;test delete flag
HL 7416           bne   uz14      ; init'd - yes
PK 7418           lda   t3        ;init start delete
FF 7420           ldy   t4        ; ptr from ptr into
GO 7422           sta   sdptr     ; program in memory
KF 7424           sty   sdptr + 1
CO 7426           sty   uzf2      ;set delete flag
KB 7428 uz14      jsr   t3bump    ;advance prg ptr
CP 7430           jmp   uz7       ;new line from disk
IA 7432 uz15      lda   linflg    ;test lines in bfr
JN 7434           beq   uz16      ;no
LH 7436           jsr   dlz       ;merge lines
IH 7438 uz16      jsr   compar    ;test dsk line # <
JI 7440           bcs   uz17      ; prg line # - no
IH 7442           jsr   t3bump    ;advance prg ptr
GN 7444           clc             ;loop
DO 7446           bcc   uz16
DE 7448 uz17      lda   t5        ;save buffer ptr
CK 7450           ldy   t6
AO 7452           sta   $22
FE 7454           sty   $23
GM 7456           jsr   makbuf    ;create new buffer
JA 7458           ldy   #$ff      ;copy most recent
CM 7460 uz18      iny             ; line into new
DJ 7462           lda   ($22),y   ; buffer
PL 7464           sta   (t5),y
IN 7466           cpy   linlen
KC 7468           bne   uz18
NC 7470           beq   uz13
GK 7472 ;
IK 7474 dskchk    ldx   #$62      ;get error channel
NK 7476           jsr   chkin     ; byte
IF 7478           jsr   prgget
DG 7480           cmp   #"2"      ;test err # < 20
GA 7482           bcc   dkc2      ;yes
PI 7484           cmp   #"7"      ;test err # = 73
LO 7486           bne   dkc1      ;no
CG 7488           jsr   prgget
HH 7490           cmp   #"3"
BP 7492           bne   dkc1      ;no
BC 7494           clc             ;flag - ok
NG 7496           .byte $24       ;'bit'
GI 7498 dkc1      sec             ;flag - error
EN 7500 dkc2      php             ;push flag
BL 7502 dkc3      jsr   prgget    ;get err msg byte
GP 7504           lda   $90       ;test end of msg
MB 7506           beq   dkc3      ;no
OO 7508           lda   #0        ;clear status byte
CC 7510           sta   $90
NC 7512           plp             ;pull flag
OL 7514           bcs   pgg1      ;quit
IE 7516           rts
EN 7518 ;
AF 7520 prgget    jsr   getin     ;get disk byte
ON 7522           pha
HN 7524           lda   $90       ;test status error
BI 7526           and   #$bf      ; except 'eoi'
JG 7528           bne   pgg1      ;yes
CP 7530           pla
IF 7532           rts
FF 7534 pgg1      jsr   uz11      ;close files, clear
KH 7536           lda   #<uzerr   ;'merge error'
GD 7538           sta   $22
HK 7540           lda   #>uzerr
IH 7542           jmp   $a445
OO 7544 ;
IO 7546 uzerr     .asc "mergE"
CP 7548 ;
IM 7550 linflg    .byte 0
CO 7552 linlen    .byte 0
KN 7554 uzf1      .byte 0         ;disk prg end flg
FJ 7556 uzf2      .byte 0         ;lines to del flg
OB 7558 uzf3      .byte 0         ;update line 95 flg
OP 7560 ;
JI 7562 getlin    ldy   #0        ;get line from
GC 7564 gtl1      sty   t2        ; disk, store from
FN 7566           jsr   prgget    ; first free byte
EJ 7568           ldy   t2        ; in buffer
JC 7570           sta   (t5),y
PF 7572           ldx   $90       ;status to disk prg
DE 7574           stx   uzf1      ; end flag
BM 7576           bne   gtl4      ;end of disk prg
MI 7578           cpy   #4        ;test link, line #
MJ 7580           bcc   gtl3      ;yes
MJ 7582           tax             ;test end of line
CO 7584           beq   gtl4      ;yes
OO 7586 gtl2      iny             ;get another byte
NL 7588           bne   gtl1
MH 7590           jsr   uz11      ;illegal 256th byte
HF 7592           jmp   $ab66     ;'file data error'
LJ 7594 gtl3      cpy   #1        ;test link hi-byte
NI 7596           bne   gtl2      ;no
FL 7598           tax             ;test link = 0
BJ 7600           bne   gtl2      ;no
NC 7602           dex             ;set disk prg end
LE 7604           stx   uzf1      ; flag
KA 7606 gtl4      sty   linlen    ;save line length
EK 7608           rts
AD 7610 ;
IJ 7612 makbuf    clc             ;put buffer half
OD 7614           lda   $37       ; way between start
FH 7616           adc   $2d       ; of variables and
CK 7618           pha             ; end of basic
OF 7620           lda   $38
HH 7622           adc   $2e
BK 7624           lsr
DK 7626           sta   siptr + 1 ;buffer start
OE 7628           sta   t6        ;buffer pointer
GF 7630           pla
EI 7632           ror
PO 7634           sta   siptr
ID 7636           sta   t5
MB 7638           lda   #0        ;clear buffer-used
EK 7640           sta   linflg    ; flag
GM 7642           rts
CF 7644 ;
CL 7646 updabp    sec             ;advance buffer pt
CG 7648           lda   linlen    ; by length of most
JA 7650           adc   t5        ; recent line
IE 7652           sta   t5
BN 7654           lda   #0
CA 7656           adc   t6
AF 7658           sta   t6
IN 7660           rts
EG 7662 ;
EC 7664 compar    ldy   #1        ;test memory-prg
PC 7666           sec             ; ptr at end of prg
```

```
HE 7668          lda  (t3),y
BI 7670          beq  com2      ;yes - sec & exit
NC 7672          ldy  #3        ;set carry if
OM 7674          lda  (t3),y    ; current line # of
HL 7676          cmp  (t5),y    ; prg in memory >
AD 7678          bcc  com1      ; line # from disk
OA 7680          bne  com1
NL 7682          dey
HF 7684          lda  (t3),y
EK 7686          cmp  (t5),y
MM 7688 com1     rts
PP 7690 com2     lda  #1        ;clear processor
IL 7692          rts            ; z flag
EI 7694 ;
FK 7696 t3bump   ldy  #4        ;advance pointer
CN 7698 mku1     lda  (t3),y    ; into program in
GC 7700          beq  mku2      ; memory by length
GI 7702          iny            ; of current line
AE 7704          bne  mku1
NO 7706          jmp  pgg1      ;illegal 256th byte
EN 7708 mku2     tya
PJ 7710          sec
ED 7712          adc  t3
CI 7714          sta  t3
IE 7716          lda  t4
DA 7718          adc  #0
KI 7720          sta  t4
GB 7722          rts
CK 7724 ;
CB 7726 chkl2    ldy  #2        ;test if current
EG 7728          lda  (t5),y    ; line from disk is
JL 7730          cmp  #2        ; line #2
FN 7732          bne  c2l4      ;no
GB 7734          iny
BJ 7736          lda  (t5),y
LN 7738          bne  c2l4      ;no
KC 7740          lda  t5        ;set chrget pointer
FL 7742          ldy  t6        ; to start of line
FK 7744          adc  #2        ; in buffer
ND 7746          sta  $7a
MO 7748          bcc  c2l1
GC 7750          iny
DC 7752 c2l1     sty  $7b
JI 7754          jsr  $73       ;get first byte
NA 7756          cmp  #$8f      ;test 'rem'
LL 7758          bne  c2l4      ;no - exit
PF 7760 c2l2     jsr  rdnum     ;get # new stmts
FN 7762          bcs  c2l4      ;not a # - exit
AE 7764          sty  stmctr
CK 7766 c2l3     cmp  #","      ;scan for comma
MC 7768          beq  c2l5      ;found
KF 7770          jsr  $73
FD 7772          bne  c2l3
KM 7774 c2l4     rts
OE 7776 c2l5     jsr  rdnum     ;get # new funcs
FO 7778          bcs  c2l4      ;not a # - exit
IE 7780          sty  functr
IB 7782          inc  uzf3      ;set update line
LF 7784          rts            ; 95 flag
AO 7786 ;
AF 7788 rdnum    jsr  $73       ;get a byte
IK 7790          bcs  rdn2      ;not a digit - exit
GO 7792          and  #$0f      ;ascii to bcd
AK 7794          tay            ;save first digit
LI 7796          jsr  $73       ;get a byte
MK 7798          bcs  rdn1      ;not a digit - exit
OO 7800          and  #$0f      ;ascii to bcd
IK 7802          sta  $22       ;pack bcd into
HD 7804          tya            ; one byte
JC 7806          asl
LC 7808          asl
NC 7810          asl
PC 7812          asl
PM 7814          ora  $22
AP 7816          tay            ;return # in .y
DC 7818          jsr  $73       ;get next byte
OM 7820 rdn1     clc            ;flag # valid
EF 7822 rdn2     rts
GA 7824 ;
HG 7826 chkl95   lda  uzf3      ;test line 95
EK 7828          beq  rdn2      ; needs update - no
HA 7830          lda  #$5f      ;search for line 95
DO 7832          ldy  #0
CG 7834          sta  $14
HM 7836          sty  $15
LO 7838          sty  uzf3      ; (clear flag)
GP 7840          jsr  $a533     ; (rechain)
DF 7842          jsr  $a613     ; (search)
PN 7844          bcc  rdn2      ;not found
KE 7846          lda  $5f       ;line 95 address to
DK 7848          ldy  $60       ; memory prg ptr,
LE 7850          sta  t3        ; and delete start
PC 7852          sty  t4        ; pointer
BM 7854          sta  sdptr
KA 7856          sty  sdptr + 1
EN 7858          adc  #2        ;set chrget ptr to
CG 7860          sta  $7a       ; start of line 95
EM 7862          bcc  c95l1
IJ 7864          iny
EA 7866 c95l1    sty  $7b
LP 7868          jsr  $73       ;get first byte
CE 7870          cmp  #"x"      ;test = 'x' for xtra
OF 7872          bne  rdn2      ;no - exit
EM 7874 c95l2    jsr  $73       ;get a byte
IL 7876          beq  rdn2      ;end of line - exit
KK 7878          bcs  c95l2     ;not a digit
CP 7880          lda  $7a       ;back up cg ptr
DA 7882          bne  c95l3
HI 7884          dec  $7b
JJ 7886 c95l3    dec  $7a
NG 7888          lda  stmctr    ;save # new stmts,
DF 7890          sta  $24       ; functions
KB 7892          lda  functr
DK 7894          sta  $25
AE 7896          jsr  c2l2      ;get # stmts, funcs
BF 7898          lda  uzf3      ;test update req'd
PI 7900          beq  rdn2      ;no - exit
HH 7902          sei
FG 7904          sed
IF 7906          clc
HH 7908          lda  stmctr    ;calc new stmt,
IM 7910          adc  $24       ; func totals
EH 7912          sta  stmctr
JK 7914          bcs  c95l4     ;>99, unreasonable
CD 7916          lda  functr
PG 7918          adc  $25
EH 7920          sta  functr
IO 7922 c95l4    cld
CI 7924          cli
LB 7926          bcs  rdn2
IL 7928          jsr  makbuf    ;create buffer
KH 7930          ldy  #0        ;create new
GM 7932 c95l5    lda  l95txt,y  ; line 95 in
HG 7934          sta  (t5),y    ; buffer
AO 7936          iny
PO 7938          cpy  #$0f
PD 7940          bne  c95l5
BD 7942          lda  stmctr    ;incorporate new
OL 7944          jsr  l95put    ; totals
FO 7946          lda  #","
DK 7948          sta  (t5),y
OO 7950          iny
GF 7952          lda  functr
BI 7954          jsr  l95put
GC 7956 c95l6    lda  l95txt,y
NK 7958          sta  (t5),y
IP 7960          iny
FO 7962          tax
IF 7964          bne  c95l6
CC 7966          dey            ;set line length
KP 7968          sty  linlen
IP 7970          sty  uzf2      ;set delete flg
KI 7972          jsr  t3bump    ;advance prg ptr
AA 7974          jsr  updabp    ;advance buffer ptr
KI 7976          jmp  dlz       ;merge new line 95
AK 7978 ;
FD 7980 l95put   pha            ;packed bcd to
AO 7982          and  #$f0      ; ascii, hi byte
FJ 7984          beq  l95p1
LA 7986          lsr
NA 7988          lsr
PA 7990          lsr
BB 7992          lsr
DA 7994          ora  #$30
JA 7996          sta  (t5),y    ;store to new line
OB 7998          iny
LH 8000 l95p1    pla            ;low byte
FF 8002          and  #$0f
NA 8004          ora  #$30
DB 8006          sta  (t5),y    ;store to new line
IC 8008          iny
GD 8010          rts
CM 8012 ;
BP 8014 stmctr   .byte 0
AO 8016 functr   .byte 0
IM 8018 ;
KN 8020 l95txt   .byte 1,1,95,0
IK 8022          .asc "xtra"
NB 8024          .byte 32
GC 8026          .asc ".byte"
EI 8028          .byte 32,32,32,32,32,32,32,59,32
MO 8030          .asc "stmts,funcs"
EF 8032          .byte 0
IN 8034 ;
PJ 8036 dlz      lda  uzf2
GJ 8038          bne  dlz1
KI 8040          lda  t3
OO 8042          ldy  t4
PH 8044          sta  sdptr
IM 8046          sty  sdptr + 1
AG 8048 dlz1     lda  #0
LL 8050          sta  uzf2
KO 8052 ;
KD 8054 delins   lda  t3        ;set up start of
LK 8056          ldy  t4        ; move
MF 8058          sta  mvstrt
AI 8060          sty  mvstrt + 1
NM 8062          sec            ;calc # bytes to
NM 8064          sbc  sdptr     ; delete
MJ 8066          sta  $22       ; $22/23
LD 8068          tya
CF 8070          sbc  sdptr + 1
PE 8072          sta  $23
DI 8074          sec            ;calc (# bytes to
HP 8076          lda  t5        ; insert) minus (#
HC 8078          sbc  siptr     ; bytes to delete)
MA 8080          pha
KL 8082          lda  t6
KG 8084          sbc  siptr + 1
FG 8086          tay
AC 8088          pla
LB 8090          sec
CD 8092          sbc  $22
CG 8094          sta  $22
HF 8096          tya
LD 8098          sbc  $23
LG 8100          sta  $23
OL 8102          clc            ;add result to
AJ 8104          lda  t3        ; prg-in-mem ptr
KC 8106          adc  $22       ; to yield move
JA 8108          sta  t3        ; destination addr
IC 8110          sta  mvdest
EN 8112          lda  t4
NC 8114          adc  $23
GB 8116          sta  t4
CP 8118          sta  mvdest + 1
EL 8120          clc            ;add same result
NH 8122          lda  $2d       ; to start-of-vars
HJ 8124          sta  mvend     ; ptr to yield move
AN 8126          adc  $22       ; end address and
PL 8128          sta  $2d       ; new start-of-vars
BI 8130          lda  $2e
ML 8132          sta  mvend + 1
BE 8134          adc  $23
FM 8136          sta  $2e
LN 8138          jsr  mvmem     ;move prg in memory
LL 8140          lda  sdptr     ;get dest addr for
AL 8142          sta  mvdest    ; new lines
MI 8144          lda  sdptr + 1
OA 8146          sta  mvdest + 1
IP 8148          lda  siptr     ;buffer addr is
AL 8150          ldy  siptr + 1 ; start of move
KL 8152          sta  mvstrt
ON 8154          sty  mvstrt + 1
OB 8156          lda  t5        ;buffer pointer
EJ 8158          ldy  t6        ; is end of move
PP 8160          sta  mvend
KD 8162          sty  mvend + 1
AO 8164          jmp  mvmem
MF 8166 ;
AE 8168 sdptr    .word 0
BF 8170 siptr    .word 0
CG 8172 ;
KE 8250 mvmem    lda  mvend     ;memory move front
LH 8252          bne  mvm1      ; end - end addr
BJ 8254          dec  mvend + 1 ; is 1 beyond
CH 8256 mvm1     dec  mvend     ; block to move
IL 8258 ;
GI 8260 mvm2     lda  mvstrt    ;set up pointer
JA 8262          sta  $22       ; low bytes
EI 8264          lda  mvdest
EB 8266          sta  $24
NC 8268          lda  mvend + 1 ;test if any bytes
IO 8270          cmp  mvstrt + 1 ; bytes to move
BC 8272          bcc  mvm5      ;no
II 8274          bne  mvm3      ;yes
FD 8276          lda  mvend
PD 8278          cmp  mvstrt
JC 8280          bcc  mvm5      ;no
KO 8282 mvm3     lda  mvdest + 1 ;test moving up
HA 8284          cmp  mvstrt + 1
CO 8286          bcc  dmvmem    ;no
GO 8288          bne  umvmem    ;yes
```

```
OO 8290             lda  $24
PC 8292             cmp $22
BG 8294             beq  mvm5     ;no move at all
NA 8296             bcc  dmvmem ;moving down
OE 8298 umvmem lda  mvend     ;init index with
KM 8300             sec               ; partial block
EG 8302             sbc  mvstrt     ; to move
PD 8304             tay
ON 8306             lda  mvend + 1;push whole blocks
AH 8308             sbc  mvstrt + 1; to move
CP 8310             pha
EE 8312             clc               ;set up pointer
GK 8314             adc  mvstrt + 1; high bytes
DE 8316             sta  $23
GA 8318             pla
GP 8320             clc
CH 8322             adc  mvdest + 1
BF 8324             sta  $25
NH 8326 mvm4   lda  ($22),y   ;perform move
HH 8328             sta  ($24),y    ; from end of
NP 8330             dey               ; block
LL 8332             cpy  #$ff
AM 8334             bne  mvm4
FH 8336             lda  $23        ;test if finished
ND 8338             cmp mvstrt + 1
HO 8340             beq  mvm5     ;yes
HO 8342             dec  $23        ;point 1 page lower
CC 8344             dec  $25
DJ 8346             jmp  mvm4     ;move another block
NI 8348 mvm5   rts
ML 8350 dmvmem lda  mvstrt + 1;set up ptr
JH 8352             sta  $23        ; high bytes
AK 8354             lda  mvdest + 1
BH 8356             sta  $25
EJ 8358             lda  mvend    ;init counter with
BO 8360             sec               ; part block size
OF 8362             sbc  mvstrt
KA 8364             sta  t2
OO 8366             inc  t2
HL 8368             lda  mvend + 1.x counts whole
KG 8370             sbc  mvstrt + 1; blocks to move
JN 8372             ldy  #0         ;init index
BI 8374             tax
LA 8376             beq  mvm7
IL 8378 mvm6   lda  ($22),y   ;move whole
EP 8380             sta  ($24),y    ; blocks, working
MM 8382             iny               ; upwards
KP 8384             bne  mvm6
LG 8386             inc  $23
DH 8388             inc  $25
NH 8390             dex
CA 8392             bne  mvm6
GL 8394 mvm7   lda  ($22),y   ;move part block
EN 8396             sta  ($24),y
OK 8398             iny
CG 8400             cpy  t2
AB 8402             bne  mvm7
AM 8404             rts
ME 8406 ;
GB 8408 mvstrt    .word 0         ;memory move start
LL 8410 mvdest    .word 0         ; . . .destination
KI 8412 mvend     .word 0         ; . . .end
EF 8414 ;
HI 8738 device    .byte 8         ;current disk #
EE 9150 errpgm    ldx  $3a        ;test hi byte of
EJ 9152             inx               ; 'curlin' = $ff
GI 9154             bne  epg1       ;no
AL 9156             rts
CA 9158 epg1      jmp  $af08      ;'syntax error'
OD 9160 ;
```

Program 2: MOVE & FILL

```
FE 0 rem move & fill (june 18/85)      :
FH 1 :
DH 2 rem 2 statements, 0 functions
HH 3 :
IO 4 rem keyword characters: 8
JH 5 :
NJ 6 rem keyword     routine     line     ser #
JI 7 rem move        mov         8174     118
OM 8 rem fill        stuf        8504     119
NH 9 :
HN 10 rem u/mvmem (8250/120)
DL 11 rem u/memfil (8416/121)
AI 12 :
LD 13 rem ================================
CI 14 :
KM 133 .asc "movEfilL"
LD 1133 .word mov-1,stuf-1
NC 8174 mov       jsr  $ad8a      ;eval and store
EN 8176             jsr  $b7f7      ; start address
GJ 8178             sta  mvstrt + 1
GD 8180             sty  mvstrt
AO 8182             jsr  $79        ;push separator
EH 8184             pha
KP 8186             jsr  $73
PJ 8188             jsr  $ad8a      ;eval and push
FI 8190             jsr  $b7f7      ; 2nd parameter
MH 8192             pha
JL 8194             tya
AI 8196             pha
IP 8198             jsr  $aefd      ;check for comma
HJ 8200             jsr  $ad8a      ;eval and store
HF 8202             sta  mvdest + 1; destination addr
GO 8204             sty  mvdest
MH 8206             pla               ;2nd parameter to
CB 8208             tay               ; .x/.y
KJ 8210             pla
PN 8212             tax
PC 8214             pla               ;test separator
GM 8216             cmp #";"        ; semicolon
PD 8218             beq  mvc1       ;yes
OJ 8220             stx  mvend + 1 ;store end address
NJ 8222             sty  mvend
BN 8224             cmp #","        ;test separator
EC 8226             beq  mvm2       ;comma - move mem
JJ 8228             jmp  $af08      ;'syntax error'
JG 8230 mvc1      clc               ;add # of bytes to
AL 8232             tya               ; move to start
CI 8234             adc  mvstrt     ; address, store as
GD 8236             sta  mvend      ; end address
CO 8238             txa
II 8240             adc  mvstrt + 1
KC 8242             sta  mvend + 1
KP 8244             bcc  mvmem      ;move memory
AP 8246             jmp  $b248      ;o'flow - 'iq err'
OK 8248 ;
KE 8250 mvmem     lda  mvend      ;memory move front
LH 8252             bne  mvm1       ; end - end addr
BJ 8254             dec  mvend + 1 ; is 1 beyond
CH 8256 mvm1      dec  mvend      ; block to move
IL 8258 ;
GI 8260 mvm2      lda  mvstrt     ;set up pointer
JA 8262             sta  $22        ; low bytes
```

```
EI 8264            lda  mvdest
EB 8266            sta  $24
NC 8268            lda  mvend+1 ;test if any bytes
IO 8270            cmp  mvstrt+1 ; bytes to move
BC 8272            bcc  mvm5    ;no
II 8274            bne  mvm3    ;yes
FD 8276            lda  mvend
PD 8278            cmp  mvstrt
JC 8280            bcc  mvm5    ;no
KO 8282 mvm3       lda  mvdest+1 ;test moving up
HA 8284            cmp  mvstrt+1
CO 8286            bcc  dmvmem  ;no
GO 8288            bne  umvmem  ;yes
OO 8290            lda  $24
PC 8292            cmp  $22
BG 8294            beq  mvm5    ;no move at all
NA 8296            bcc  dmvmem  ;moving down
OE 8298 umvmem     lda  mvend   ;init index with
KM 8300            sec          ; partial block
EG 8302            sbc  mvstrt  ; to move
PD 8304            tay
ON 8306            lda  mvend+1 ;push whole blocks
AH 8308            sbc  mvstrt+1 ; to move
CP 8310            pha
EE 8312            clc          ;set up pointer
GK 8314            adc  mvstrt+1 ; high bytes
DE 8316            sta  $23
GA 8318            pla
GP 8320            clc
CH 8322            adc  mvdest+1
BF 8324            sta  $25
NH 8326 mvm4       lda  ($22),y ;perform move
HH 8328            sta  ($24),y ; from end of
NP 8330            dey          ; block
LL 8332            cpy  #$ff
AM 8334            bne  mvm4
FH 8336            lda  $23     ;test if finished
ND 8338            cmp  mvstrt+1
HO 8340            beq  mvm5    ;yes
HO 8342            dec  $23     ;point 1 page lower
CC 8344            dec  $25
DJ 8346            jmp  mvm4    ;move another block
NI 8348 mvm5       rts
ML 8350 dmvmem     lda  mvstrt+1 ;set up ptr
JH 8352            sta  $23     ; high bytes
AK 8354            lda  mvdest+1
BH 8356            sta  $25
EJ 8358            lda  mvend   ;init counter with
BO 8360            sec          ; part block size
OF 8362            sbc  mvstrt
KA 8364            sta  t2
OO 8366            inc  t2
HL 8368            lda  mvend+1 ;.x counts whole
KG 8370            sbc  mvstrt+1 ; blocks to move
JN 8372            ldy  #0      ;init index
BI 8374            tax
LA 8376            beq  mvm7
IL 8378 mvm6       lda  ($22),y ;move whole
EP 8380            sta  ($24),y ; blocks, working
MM 8382            iny          ; upwards
KP 8384            bne  mvm6
LG 8386            inc  $23
DH 8388            inc  $25
NH 8390            dex
CA 8392            bne  mvm6
GL 8394 mvm7       lda  ($22),y ;move part block
EN 8396            sta  ($24),y
OK 8398            iny
CG 8400            cpy  t2
AB 8402            bne  mvm7
AM 8404            rts
ME 8406 ;
GB 8408 mvstrt     .word 0      ;memory move start
LL 8410 mvdest     .word 0      ; . . .destination
KI 8412 mvend      .word 0      ; . . .end
EF 8414 ;
FH 8416 memfil     clc          ;start + bytes flag
HA 8418            .byte $24    ;'bit'
EA 8420 memf1      sec          ;start, end flag
PP 8422            sty  t3      ;store start addr
KE 8424            sta  t4
PG 8426            bcc  memf3   ;skip calc
NJ 8428 memf2      lda  $14     ;calc bytes to fill
PM 8430            sbc  t3      ; - end minus start
IL 8432            sta  $14
PH 8434            lda  $15
IC 8436            sbc  t4
BM 8438            sta  $15
KH 8440            bcc  memf7   ;end < start
MB 8442            inc  $14     ;bump bytes to fill
HI 8444            bne  memf3
LK 8446            inc  $15
LC 8448 memf3      lda  $14     ;test fill area
GL 8450            clc          ; in memory
IB 8452            adc  t3
FN 8454            tay
FJ 8456            lda  $15
AC 8458            adc  t4
HF 8460            bcc  memf4   ;yes
KD 8462            bne  memf8   ;no
HM 8464            tya
OD 8466            bne  memf8   ;no
EA 8468 memf4      txa          ;get fill character
LD 8470            ldy  #0      ;init index
CA 8472            ldx  $15     ;# blocks to fill
GK 8474            beq  memf6   ;none
NC 8476 memf5      sta  (t3),y  ;fill a block
OP 8478            iny
NK 8480            bne  memf5
GG 8482            inc  t4
CH 8484            dex          ;test more to fill
LJ 8486            bne  memf5   ;yes
CI 8488 memf6      cpy  $14     ;# bytes to fill
HL 8490            beq  memf7   ;none
DF 8492            sta  (t3),y  ;fill a byte
OA 8494            iny
OL 8496            bne  memf6
MB 8498 memf7      rts
FP 8500 memf8      jmp  $b248   ;'illegal qty'
MK 8502 ;
GH 8504 stuf       jsr  $ad8a   ;push start address
LK 8506            jsr  $b7f7
IL 8508            pha
FP 8510            tya
ML 8512            pha
MC 8514            jsr  $79     ;push separator
AM 8516            pha
FJ 8518            jsr  $73     ;eval 2nd paramater
```

```
AL | 8520          jsr  $ad8a
LL | 8522          jsr  $b7f7
HH | 8524          jsr  $79        ;test for 3rd param
EM | 8526          beq  stuf1      ;no
CE | 8528          jsr  $aefd      ;check for comma
FK | 8530          jsr  $b79e      ;get fill character
 I | 8532          .byte $2c       ;'bit'
DC | 8534 stuf1    ldx  #0         ;fill with 0
AJ | 8536          pla             ;separator to .y
FH | 8538          tay             ;start address to
AN | 8540          pla             ; t3/t4
OL | 8542          sta  t3
IO | 8544          pla
EM | 8546          sta  t4
DI | 8548          cpy  #","       ;comma separator
FF | 8550          beq  memf2      ;fill
DM | 8552          cpy  #";"       ;semicolon sep'r
KF | 8554          beq  memf3      ;fill
BO | 8556          jmp  $af08      ;'syntax error'
EO | 8558 ;
```

Program 3: DOS SUPPORT

```
HJ | 0 rem dos support (d. spruyt, 1985)  :
FH | 1 :
CI | 2 rem 5 statements, 2 functions
HH | 3 :
LJ | 4 rem keyword chars: 24
JH | 5 :
NJ | 6 rem  keyword   routine   line    ser #
KF | 7 rem  s/cat     kat       8644    123
JB | 8 rem  s/dos     comms     8742    124
OE | 9 rem  s/dev     dvc       8766    125
JL | 10 rem s/dload    dld       8808    126
KC | 11 rem s/dsave    dsve      8814    127
PM | 12 rem f/ds$      dss       8598    128
DP | 13 rem f/ds       dsn       8618    129
CI | 14 :
BF | 15 rem u/usfp (2620/006)
EI | 16 :
PD | 17 rem ================================
GI | 18 :
MP | 134 .asc "caTdoSdeV"
JB | 135 .asc "dloaDdsavE"
HM | 620 .asc "ds":.byte $a4:.asc "dS"
HH | 1134 .word kat-1,comms-1,dvc-1
MJ | 1135 .word dld-1,dsve-1
LN | 1620 .word dss-1,dsn-1
IB | 2620 usfp    ldx  #0         ; routine to convert
GM | 2622          stx  $0d        ; unsigned integer
IN | 2624          sta  $62        ; in .a (high byte)
OH | 2626          sty  $63        ; and .y (low byte)
BB | 2628          ldx  #$90       ; to floating point
FJ | 2630          sec             ; in fpa #1
NH | 2632          jmp  $bc49
AM | 2634 ;
HJ | 8560 dss1    lda  device
FH | 8562          sta  $ba        ;set device
BA | 8564          lda  #$6f
OP | 8566          sta  $b9        ;set secondary
DG | 8568          lda  #0
OJ | 8570          sta  $b7        ;filename length
JN | 8572          jsr  chpres     ;chk dev present
BF | 8574          lda  $ba
JC | 8576          jsr  $ffb4      ;send talk device
ND | 8578          lda  $b9
BC | 8580          jsr  $ff96      ;send talk scndry
DK | 8582          ldy  #$ff
HG | 8584 dss2     iny
CE | 8586          jsr  $ffa5      ;get from disk
ML | 8588          sta  dsbuf,y    ; and put into
HD | 8590          cmp  #$0d       ; buffer until
EC | 8592          bne  dss2       ; a return
PA | 8594          jsr  $f642      ;untalk device
AI | 8596          rts
PN | 8598 dss      jsr  dss1       ;get disk message
PE | 8600          tya
ID | 8602          jsr  $b47d      ;create space
LG | 8604          tay
AI | 8606 dss3     lda  dsbuf,y    ;from buff
LD | 8608          sta  ($62),y    ;to memory
NF | 8610          dey
HP | 8612          bpl  dss3
NO | 8614          jmp  $b4ca      ;clean desc stack
OB | 8616 ;
FO | 8618 dsn      jsr  dss1
LL | 8620          ldx  dsbuf      ;first digit (10s)
BN | 8622          lda  dsbuf+1    ;second digit
IG | 8624          and  #$0f       ;ascii to hex
EB | 8626 dsn1     cpx  #"0"       ;add 10s
HO | 8628          beq  dsn2
NG | 8630          dex
HK | 8632          adc  #9
LK | 8634          bcc  dsn1
HL | 8636 dsn2     tay             ;convert to
CC | 8638          lda  #0         ; floating point
 I | 8640          jmp  usfp
ID | 8642 ;
JF | 8644 kat      lda  device     ;catalog f'n
JM | 8646          sta  $ba        ;set device
NP | 8648          lda  #$60
MH | 8650          sta  $b9        ;set sncdary
JL | 8652          lda  #1
CB | 8654          ldy  #>dollar
EB | 8656          ldx  #<dollar
AP | 8658          jsr  $ffbd      ;set string
HB | 8660          jsr  $f3d5      ;send sa + string
JK | 8662          lda  $ba
AM | 8664          jsr  $ffb4      ;send talk
FJ | 8666          lda  $b9
HH | 8668          jsr  $ff96      ;send talk sa
JM | 8670          lda  #0
KD | 8672          sta  $90        ;clear st
ND | 8674          jsr  $ffa5      ;discard load add
EH | 8676          jsr  $ffa5
EM | 8678 kat1     jsr  $ffa5      ;discard line link
IH | 8680          jsr  $ffa5
AP | 8682          ldx  $90        ;test status
PA | 8684          bne  kat2       ;end of file
MG | 8686          tax             ;test link hi
CB | 8688          bne  kat3       ;not end
JI | 8690 kat2     jsr  $f642      ;untalk
GD | 8692          jmp  $aad7      ;print <cr>, exit
OF | 8694 kat3     jsr  $aad7      ;print <cr>
KM | 8696          jsr  $ffa5      ;get 'line #'
JK | 8698          sta  t2         ; (file size)
MI | 8700          jsr  $ffa5
```

```
KH 8702          ldx  t2
CF 8704          jsr  $bdcd      ;print it
BJ 8706          jsr  $ab3f      ;print space
AL 8708 kat4     jsr  $ffa5      ;get next char
AP 8710          ldx  $90
KO 8712          bne  kat2       ;check st
AL 8714          jsr  $ffd2      ;print char
EK 8716          bne  kat4       ;not end of line
OP 8718          jsr  $ffe1      ;test stop key
LO 8720          beq  kat2       ;yes - end
MH 8722          jsr  $ffe4      ;getin
LD 8724          beq  kat1       ;no keypress
EO 8726 kat5     jsr  $ffe1      ;test stop key
IE 8728          beq  kat2       ;yes
EI 8730          jsr  $ffe4      ;getin
DF 8732          beq  kat5       ;no keypress
FD 8734          bne  kat1       ;yes
FB 8736 dollar   .byte "$"
DG 8738 device   .byte 8
KJ 8740 ;
OB 8742 comms    jsr  $ad9e      ;eval exp
HB 8744          jsr  $b6a3      ;clr desc + check str
AJ 8746          sta  $b7        ;save length
CM 8748          sty  $bc        ;save pntr to it
OJ 8750          stx  $bb
JF 8752          lda  device
FD 8754          sta  $ba        ;set device
BM 8756          lda  #$6f
OL 8758          sta  $b9        ;set secondary
FJ 8760          jsr  chpres     ;chk dev present
DG 8762          jmp  $f3d5      ;send sa + string
CL 8764 ;
FO 8766 dvc      jsr  $b79e      ;get param
BB 8768          cpx  #8         ;test 8-11
AO 8770          bcc  dvc2       ;no
BC 8772          cpx  #$0c
EC 8774          bcs  dvc2       ;no
NL 8776          stx  device     ;set device #
NP 8778 dvc1     rts
DJ 8780 dvc2     jmp  $b248      ;illegal qty
EM 8782 ;
DB 8784 name     lda  #0
KN 8786          sta  $b9        ;set secondary
PA 8788          sta  $0a        ;set load flag
DM 8790          jsr  $e257      ;eval string to mem
BI 8792          lda  device
NF 8794          sta  $ba        ;set device
PK 8796          jsr  $79        ;get char
KP 8798          beq  dvc1       ;end of statement
BG 8800          jsr  $aefd      ;chk for comma
EM 8802          jsr  $b79e      ;get device number
CJ 8804          jmp  $e1ec      ;handle setup
MN 8806 ;
FD 8808 dld      jsr  name       ;check string
DL 8810          jmp  $e16f      ;load
CO 8812 ;
FO 8814 dsve     jsr  name       ;check string
IK 8816          jmp  $e159      ;load
IO 8818 ;
BC 8820 chpres   lda  #0         ;clear status
CE 8822          sta  $90
GE 8824          lda  $ba        ;listen
KA 8826          jsr  $ffb1
IN 8828          jsr  $ffae      ;unlisten
IC 8830          lda  $90        ;test status
LB 8832          bne  chp1       ;bad
OG 8834          rts
JO 8836 chp1     ldx  #5         ;'dev not present'
MI 8838          jmp  ($300)
OP 8840 ;
CF 8842 dsbuf    *=*+$24         ;disk msg buffer
CA 8844 ;
```

Program 4: LINE CALC

```
NA 0 rem line calc (7/85)          :
FH 1 :
KC 2 rem 2 statements,  1 function
HH 3 :
HE 4 rem keyword characters: 13
JH 5 :
NJ 6 rem keyword     routine    line     ser #
JE 7 rem s/jump      jum        8846     130
MM 8 rem s/call      cal        8870     131
JH 9 rem f/line(     line       8902     132
OH 10 :
NE 11 rem u/usfp (2620/006)
AI 12 :
LD 13 rem ================================
CI 14 :
EN 136 .asc "jumPcalL"
IL 621 .asc "line": .byte $a8
HN 1136 .word jum-1,cal-1
DJ 1621 .word line-1
IB 2620 usfp     ldx  #0         ;routine to convert
GM 2622          stx  $0d        ;unsigned integer
IN 2624          sta  $62        ;in .a (high byte)
OH 2626          sty  $63        ;and .y (low byte)
BB 2628          ldx  #$90       ;to floating point
FJ 2630          sec             ;in fpa #1
NH 2632          jmp  $bc49
AM 2634 ;
IP 8846 jum      jsr  $b08b      ;find variable
HF 8848          bit  $0e        ;test integer type
FI 8850          bpl  jm1        ;no
AH 8852          ldy  #0         ;get line # lo byte
CH 8854          lda  ($47),y
JI 8856          sta  $60        ;set up for 'goto'
NM 8858          iny             ;get line # hi byte
IH 8860          lda  ($47),y
PB 8862          sec
EE 8864          jmp  $a8c7      ;enter 'goto' rtn
GH 8866 jm1      jmp  $ad99      ;'var typ mismatch'
KB 8868 ;
IP 8870 cal      lda  #3         ;test stack depth
GB 8872          jsr  $a3fb
EK 8874          lda  $7b        ;push chrget ptr
IC 8876          pha
LG 8878          lda  $7a
MC 8880          pha
FG 8882          lda  $3a        ;push line #
AD 8884          pha
DF 8886          lda  $39
ED 8888          pha
FI 8890          lda  #$8d       ;push 'gosub' token
ID 8892          pha
IG 8894          jsr  $79        ;enter 'goto' rtn
```

```
AH | 8896          jsr  jum
AN | 8898          jmp  $a7ae      ;begin subroutine
KD | 8900 ;
LJ | 8902 line    cmp  #$89       ;skip goto token
NI | 8904          bne  line1      ; if present
KM | 8906          jsr  $73
DE | 8908 line1   lda  #2         ;check stack depth
MD | 8910          jsr  $a3fb
NL | 8912          lda  $14        ;save current
JO | 8914          pha             ; values of
BK | 8916          lda  $15        ; affected memory
CF | 8918          pha
AK | 8920          lda  $5f
GF | 8922          pha
EG | 8924          lda  $60
KF | 8926          pha
AB | 8928          jsr  $ad8a      ;eval line # expr
OG | 8930          jsr  $b7f7      ;conv to integer
AD | 8932          lda  #0         ;zero fac 1
PK | 8934          sta  $61
HE | 8936          jsr  $a613      ;find line address
BP | 8938          bcc  line2      ;undef'd statement
BG | 8940          ldy  $5f        ;convert to
KC | 8942          lda  $60        ; floating point
CK | 8944 line2   jsr  usfp       ; in fac #1
AI | 8946          pla             ;restore memory
KL | 8948          sta  $60
OH | 8950          pla
OP | 8952          sta  $5f
CI | 8954          pla
HM | 8956          sta  $15
GI | 8958          pla
IM | 8960          sta  $14
GD | 8962          jmp  $aef7      ;check close paren
KH | 8964 ;
```

Program 5: BEEP

```
CG | 0 rem beep (sept 1/85)
FH | 1 :
AI | 2 rem  1 statement, 0 functions
HH | 3 :
EO | 4 rem keyword characters: 4
JH | 5 :
NJ | 6 rem keyword     routine     line      ser #
EN | 7 rem s/beep      bee         8966      133
MH | 8 :
OO | 9 rem ================================
OH | 10 :
ML | 137 .asc "beeP"
KE | 1137 .word bee-1
FK | 8966 bee    beq  bp1        ;no parameters
NC | 8968          jsr  $b79e      ;eval duration
PP | 8970          inx             ;bump duration
AE | 8972          .byte $2c       ;'bit'
LI | 8974 bp1     ldx  #1         ;default duration
JA | 8976          txa             ;push duration
OI | 8978          pha
FN | 8980          ldx  #$21       ;default pitch
NM | 8982          ldy  #$87
FP | 8984          jsr  $79        ;test pitch param
BA | 8986          beq  bp2        ;no
OA | 8988          jsr  $aefd      ;check for comma
```

```
BB | 8990          jsr  $ad8a      ;eval pitch
MK | 8992          jsr  $b7f7      ;conv to integer
NO | 8994          tax
OC | 8996 bp2     sty  $d40e      ;write pitch to sid
GH | 8998          stx  $d40f
IN | 9000          ldx  #0         ;clear attack/decay
FL | 9002          stx  $d413      ; and filter select
BH | 9004          stx  $d417
MH | 9006          lda  #$f0       ;sustain 15, rel 0
GB | 9008          sta  $d414
IG | 9010          lda  #$0f       ;volume 15
OB | 9012          sta  $d418
AO | 9014          lda  #$21       ;gate on (sawtooth)
MB | 9016          sta  $d412
PO | 9018          pla             ;pull duration
NL | 9020          sec
HA | 9022 bp3     ldy  #8         ;countdown duration
KF | 9024 bp4     dex
CE | 9026          bne  bp4
PP | 9028          dey
GE | 9030          bne  bp4
FE | 9032          sbc  #1
HE | 9034          bne  bp3
OB | 9036          lda  #$20       ;gate off
CD | 9038          sta  $d412
MD | 9040          rts
IM | 9042 ;
```

Program 6: Stripper

```
IM | 100 rem stripper
PL | 102 rem remove comments from
GA | 104 rem pal source code
ON | 106 :
NL | 108 for i=900 to i+20
IN | 110 read a: poke i,a
EH | 112 next
GO | 114 :
BC | 116 u$="/ / / / / / / / / ]": q$=chr$(34)
   |     Note: u$=(1 apostrophe + 1 space) × 8 + 1 ]
KO | 118 :
CK | 120 print "keycH(149),";q$;"s/";*]//";
OL | 122 print q$;":clr"
AP | 124 :
CB | 126 for i=150 to 153
EM | 128 print:print
IH | 130 v$=mid$(str$(i),2)
AE | 132 print "keycH(";v$;"),";q$;
LB | 134 print "s/";right$(u$,2↑(154-i)+1);
FI | 136 print "//";q$;":clr"
JG | 138 next i
AA | 140 :
IJ | 142 print "qqqr use sys 900"
EA | 144 :
LI | 146 for i=1 to 19: print "Q";: next
JL | 148 for i=631 to 635: poke i,13: next
KG | 150 poke 198,5
IJ | 152 end
OA | 154 :
BO | 156 data 160,  10, 162, 153, 132, 198
MI | 158 data 169,  13, 136, 153, 119,   2
CN | 160 data 138, 202, 136, 153, 119,   2
HE | 162 data  16, 242,  96
```

Sky Travel – A Review

Richard Evers, Editor

"Genius is the ability to reduce the complicated to the simple." - C.W. Ceram

The above statement must have been made with Frank Covitz and Clif Ashcraft (authors of Sky Travel) in mind. Never before have I been this impressed with any software package. Just thinking about the task involved in writing this magnificent piece of code leaves me in a panic. Most major software applications are difficult to write, but Sky Travel must have been close to impossible. And the trick is, Sky Travel makes it all look so easy. Beautiful!!

Before continuing with my Sky Travel review, a little story is in order.

The James Mitchener Story

For the past nine years, James Mitchener has been my favourite novelist. Although his writing style has given pleasure to millions, a greater number have yet to discover his talent. If the titles Hawaii, Tales of the South Pacific, The Source, Centennial, The Covenant, and Space fail to ring a familiar bell, then prepare for an awakening. These are but a few of the novels James Mitchener has written, written for the sole purpose of giving literary pleasure and knowledge to all. Although his writing style tends to be tedious and drawn out at the beginning of each novel, rest assured that he is only laying the correct ground work for the balance of the novel, and will soon have you entranced in his literary spell. You will become one with the story, you will be drawn into the settings, the people, and the history as it progresses. You will be able to climb inside of Mr. Mitchener's mind, and absorb all that he offers. His novels are always extremely well researched, with a presentation surpassed by few. Knowledge and pleasure. Who could ask for more?

The reason why I have skirted the main subject and introduced you to James Mitchener is because of the novel Space. This novel was for me the starting point in my Astronomical learning process. Before Space, I had little interest in Astronomy, nor felt any need for it. Thanks to James Mitchener's talent, I was given a basic understanding of Astronomy, in a manner that was pleasant to digest. And thanks to the two years he spent working at NASA, an insight was also supplied regarding the complexities involved with space travel, and the steps taken to solve the problems encountered. As a final salute, if you haven't read any of his novels, start with Space. You will not be disappointed.

Back To Sky Travel

The manual supplied with Sky Travel is probably one of the finest quick Astronomical tutorials you will ever read. Along with learning how to use the program, you will also be given many important Astronomical facts, plus just enough trivia to keep you intrigued. Front to back, the manual is a delight. Through its reading, you will learn about Longitude and Latitude, Declination and Right–Ascention (same as Longitude and Latitude but for space), and the Time Zones. Further to that you will realize why leap years exist; each year is comprised of 365.2422 days! Due to this fact, you will learn about the multitude of calendar systems used throughout the ages. One of the great pains in writing this system must have been in compensating for the calendar changes.

To further brighten my day while reading the manual, I found the term 'precession'. This is a term used to describe the slight wobble the earth experiences about its axis. This fact was presented to me while researching the translation and interpretation of the writings of Michael Nostradamus, in particular those pertaining to the "final war to end all wars in the year 1999". Through my research, I was able to disprove the 1999 theory entirely. As a bonus, I also found out about precession. This slight wobbling effect has led to some strange changes, from our viewpoint, in the cosmos. For example, everybody knows of Polaris, the North Star. It's the first star in the Little Dipper. Well, due to this wobble, Polaris has not always been the North Star, nor will it continue to be in the future. Also, the signs of the Zodiac are all slightly out of phase due to precession. This known factor has never been incorporated into the 'science' of Astrology, therefore, the sign you were born under may not be *your* sign!

A Birthday!!

To continue, another interesting fact derived from the Sky Travel manual is that Jesus Christ was born on September 15 in the year 7 BC. Now this is a miracle!! Born seven years before conception!!

As history goes, a monk in the 6th century AD put forth the idea to date the calendar from Jesus's birth, which was calculated to be in the year 754 A.U.C. (Read the manual to figure that one out.) This change was instantly accepted by Rome, but

it took another five centuries for the rest of the world to conform. Don't you love good trivia?

The date of September 15 in 7 BC was calculated after considering that King Herod died in the year 4 BC, before the recorded birth. From that point, calculations were made to find the Star of Bethlehem, as it became known. Without dragging the point too far, the only occurence that could have led to such a bright 'Star' could have been the alignment of the Stars Saturn and Jupiter on the night of September 15th in 7 BC. If the Christian religions follow known astronomical calculations, then this fact is correct. If the 'Star' was a special birthday present from the powers above, then who really knows? December 25th seems like a pretty good day for Christmas.

Without going on forever, the manual is great, and could sell as a stand alone item. But, overshadowing the manual comes a terrific program, Sky Travel.

Sky Travel: The Program

You will find, when first firing the system up, that more than a few options present themselves. Through Sky Travel, you can synthetically locate yourself accurately anywhere on Earth, at any time, any date, in any year, in the past or future by 10,000 years. From this point, you can view the cosmos as you please. You can stop time and just sit and view. You can advance the rate of time by up to a factor of 64, forwards or backwards. And you can change your screen viewing angle from 72 degrees down to 9 degrees.

The display options given are pretty impressive, as shown below:

Lines : Shows principal constellation lines when enabled

Names : Places abbreviations of the names of the constellations shown, next to the constellations when enabled.

Symbols : Will display the commonly used symbols relating to each planet, next to the planets displayed when enabled.

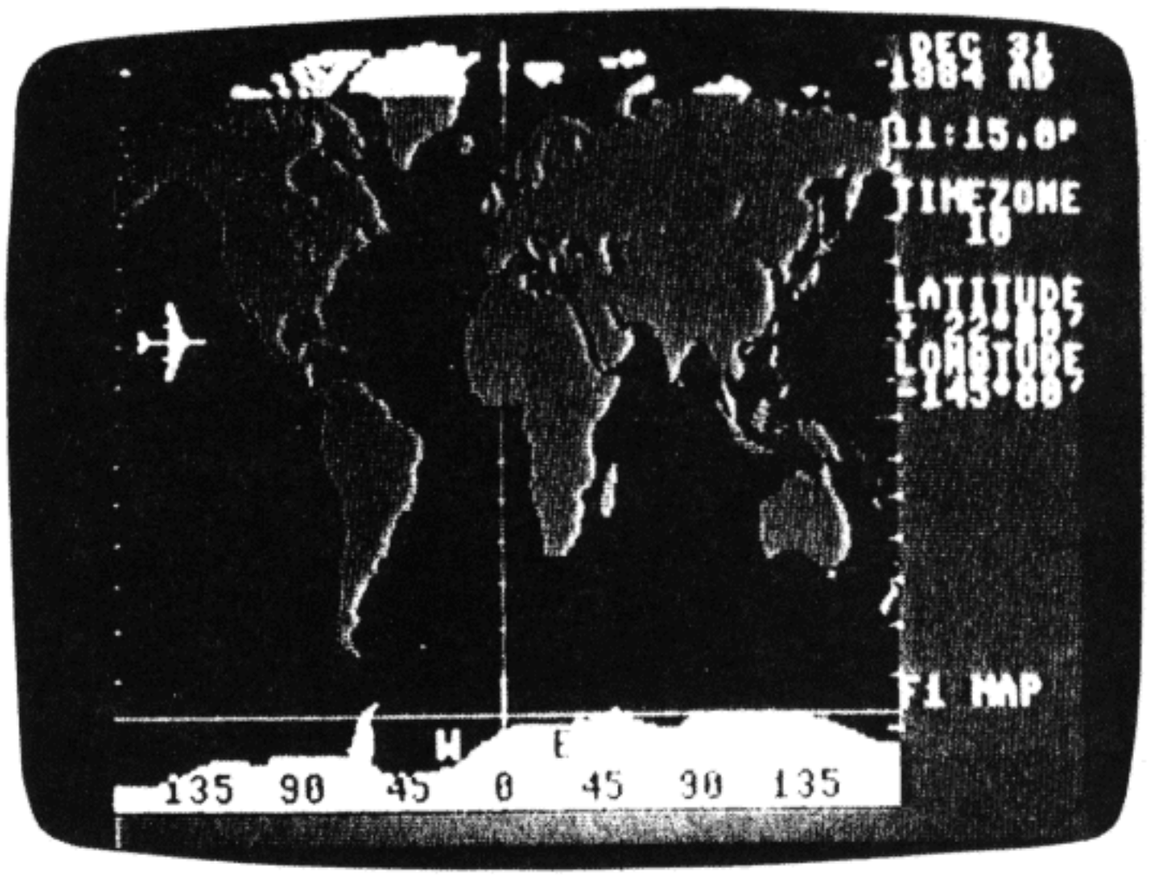
Deep Sky: Displays the distant deep space nebula and galaxies along with the normal display of our own galaxy when enabled.

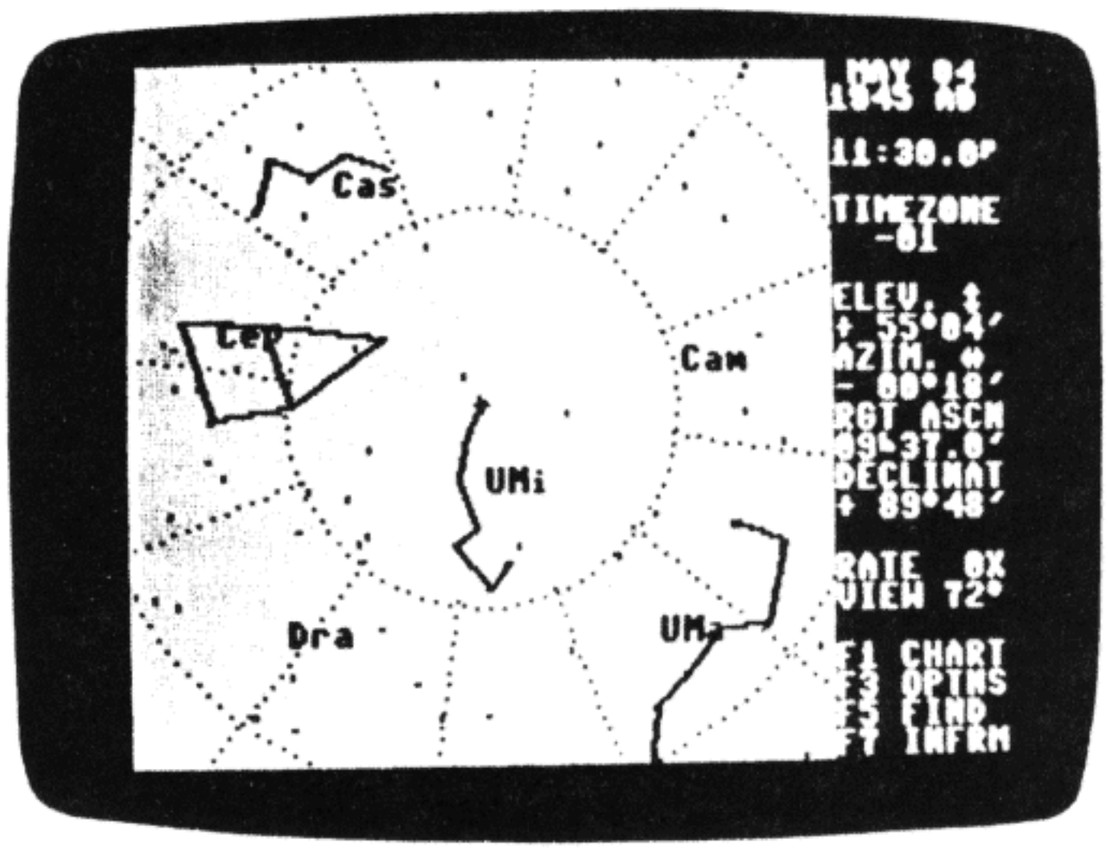
Track : Allows tracking of the Sun, Moon, the planets, or Halleys Comet when enabled. Tracking means that it will follow the desired object along its path as long as it is observable.

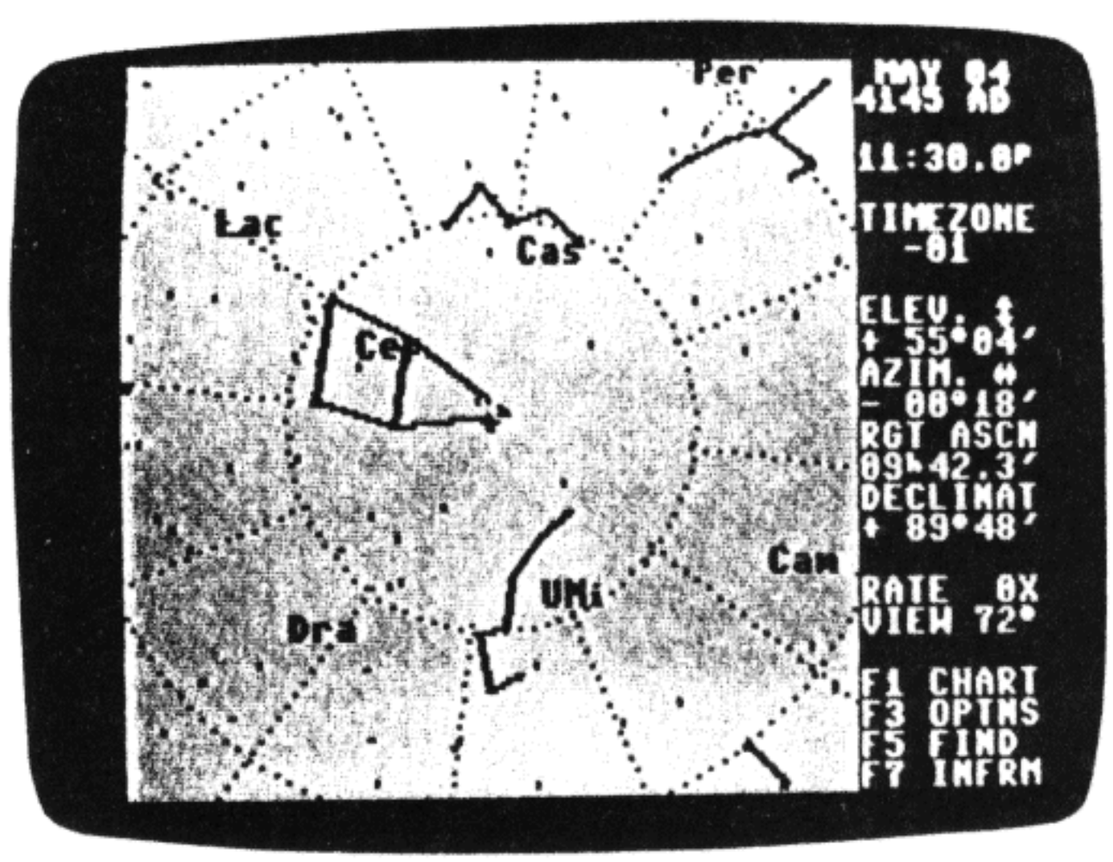
Sound : With Sound on, your cross–hairs (cursor) will be turned into a space ship on the screen, with sound effects thrown in for good measure. When in Map mode (setting your location on Earth), you will have an airplane instead of the normal cross–hairs, along with sound effects. This was implemented to encourage children to use the system.

A Few More Nice Touches

To move ahead, you also have a Find function that lets you rapidly find the Moon, Sun, any of the planets, any comet that might happen to be around, or any constellation available that appeals to you. A nice touch. As fascinating as the Find function is the Inform function. All you do is line your cross–hairs up to any object on display, then press the Inform function key. From disk will come a quick synopsis of up to date data regarding the chosen object.

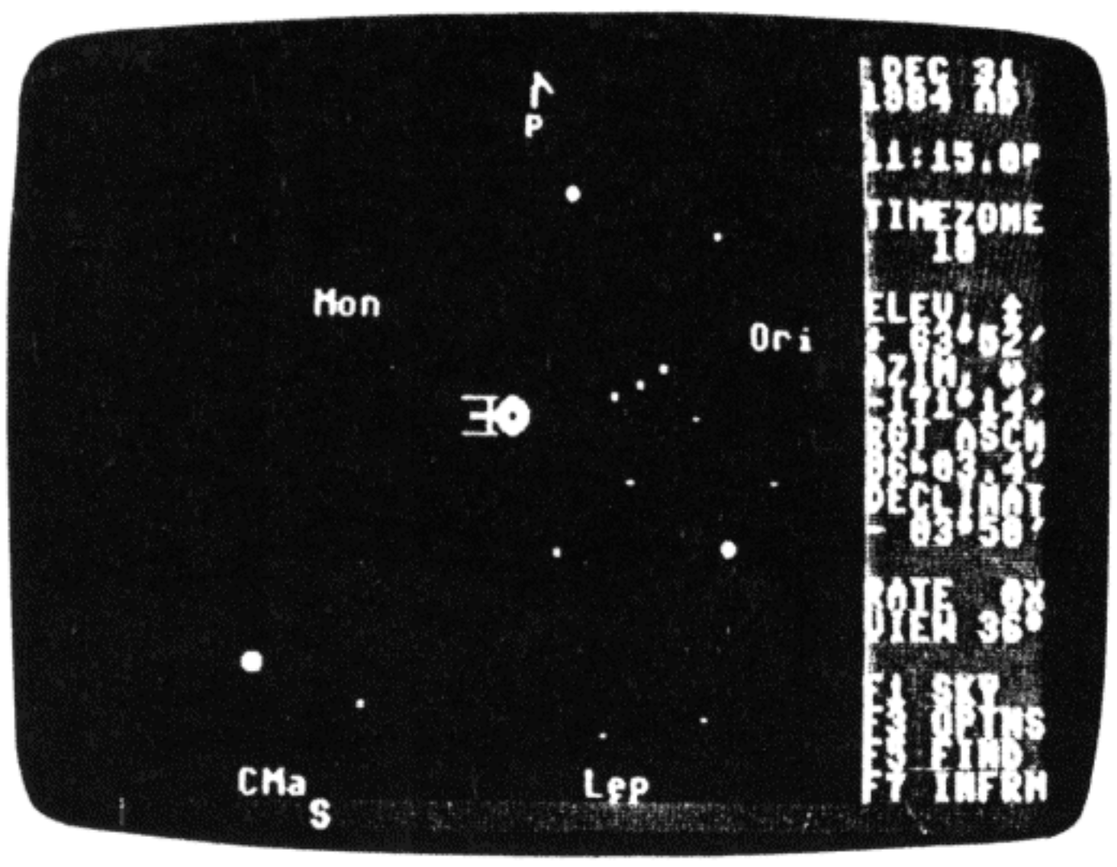






As for the actual mode of display, two modes exist. They are Sky and Chart. In Sky mode you can use the cursor keys, plus assorted other special keys, to move about the screen, and cause 'Skewing' into new areas of view. As you move the cross–hairs to indicate viewing beyond the edge of the screen, the computer skews the display to reflect this change. As you move the cross–hairs on the screen, you will notice that the Elevation, Azimuth, Right Ascention, and Declination given on the screen are updated accordingly. And, if you require a print out of the display shown, a (Shift) (p) will do the trick for any serial printer, Unit #4, on line.

In Chart mode you will notice that your display is a reverse of what you see in Sky mode. This mode is best used when printing out high resolution displays to your printer for plotting purposes. The dark objects on a white background will save your ribbon in relation to the reverse. In comparison, other than the reverse image and the fact that you can no longer skew beyond the edges of display, the Chart and Sky mode appear identical.



A Beef

Sky Travel is an incredible program, but still, I found one thing to beef about; the disk protection used. The read errors encountered during system initialization cause one heck of a lot of head banging. Not such a nice thing to hear just after getting my drive aligned!! In case you're interested, you can turn off the 1541's head bumping trick by executing the following code before loading in the program.

```
open 15,8,15: print#15, " m–w " chr$(106)
chr$(0)chr$(1)chr$(133): close 15
```

This sets bit 7 at location 106, REVCNT, tested by DOS with a BIT command when an error is encountered on disk. If the negation flag is set after the BIT, the code bypasses the head bump routine, therefore bringing instant relief to all service technicians.

The David Dunlop Observatory

Just after starting to write this review, I had the opportunity to visit the University of Toronto's David Dunlop Observatory in Richmond Hill, with my brother John. Both John and I are budding junior astronomers, and therefore we were both excited about the prospect of visiting the observatory. Although we grew up in the Hill, and I currently live in the Hill with my wife and daughter, it never occurred to us to take the time to visit the observatory. One phone call and a free lecture, tour, and peek up the scope was our reward.

Actually, we never got that peek up the scope that night. It was our luck to pick a night when the sky completely emptied on Richmond Hill. Monsoon season or something. Anyway, the facilities are pretty good, the lecture was informative, and the telescope, although 52 years old, has plenty of life left in her. Weighing in at 30 tons, this telescope looks quite impressive. If only we could have seen something through it.

During that night, we talked with a few people working at the observatory, and found that nobody was using Commodore equipment. But, once described to them, everyone seemed quite excited about the possibilities available with the package Sky Travel. Note To Commodore: Give the U of T's Faculty of Astronomy a call. Good chance of snowballing some sales, with the right approach.

In Summation

As you can guess, I feel that the program is A1. It is so well designed, and appeals to such a specific market, that it must have been written as a labour of love. I just hope that this review will convince you that Astronomy can be as exciting as you make it. With Sky Travel at your side, the universe is within your reach.

"Sky Travel" or "Shy Travel"?

As brilliant as Sky Travel is, it too is not immune to the perils of the software realm. You may find it difficult to find Sky Travel. The package is a Commodore product but it appears once more that Commodore has made a retreat from the software front. Many retailers have trimmed their inventories down to the "big movers" and have become fearful of "new dust catchers to add to their collection". But it would seem unlikely that the number of Sky Travels sold matches a typical minimum production run. For those determined enough, Commodore could probably point you towards one - they must be out there, but where? Perhaps the 'Find' command will help. We'll try it and let you know.

The latest development in the Sky Travel story is Planet Travel. Talk about stunning. When complete, this program will be the "Flight Simulator" of space travel. A demonstration sampler shows Saturn from several perspectives with the stars, sun, shadows, moons, all accurately plotted. I suspect they'll be two seperate items, though. Sky Travel already fills most of the disk so don't wait - you'll probably need both to "get it all". – M.Ed

Accurate Sum Of Squares

John Jay Hilfiger
Ithaca, NY

Another technique for avoiding wrong answers

John Jay Hilfiger is Manager of the Statistical Computing Department at Cornell University. Rest assured the following is not merely another re-hash of the binary arithmetic problem but rather an approach for dealing with it. – M.Ed

Your Commodore computer can do difficult numerical analyses with amazing speed, and is a very handy tool for this reason. You may have come to rely on your computer for important calculations, but sometimes those calculations may be very wrong! This happens because computers can only approximate some numbers. The BASIC interpreter on eight bit computers like the Commodore 64 and VIC–20 represents most numbers as nine digit approximations. This may seem like more than enough precision, and it usually is, but there are times when it gets us into trouble. Does this mean that small computers are not for serious calculations? Alas, even the biggest computers have limitations, but choosing the right algorithms and careful programming can help us to get around many of those limitations.

Many problems in statistics and other branches of science require finding the sum of the squares of a group of numbers,

$$\Sigma (x - \bar{x})^2$$

where x is any number and $\bar{x}$ is the average. This is simply the total of the squared differences between each number and the average of all of them. Consider the numbers 0, 1, 2. Their average or "mean" is 1 and the sum of the squares is 2. Program 1 finds the mean and sum of squares of this set.

Program 1 makes two passes through the data, that is, it reads all of the data twice. There is nothing wrong with this, but if we had a lot of data, Program 1 would be very slow.

Many textbooks that were written with hand or calculator computation in mind suggest the following algebraically equivalent method of finding sums of squares, requiring only one pass through the data:

$$\Sigma x^2 - (\Sigma x)^2/n$$

where n is the number of numbers in the data set. This formula is often used by programmers of statistical software because it is much faster than the two pass method and because they are unaware of the inherent dangers. Methods designed for hand calculation are not necessarily the best methods for computers to use. Program 2 implements this formula, which we shall refer to as the "calculator algorithm".

Either Program 1 or Program 2 gives the rights answers for the simple data set 0, 1, 2. Let us try a more difficult set of numbers, 30000, 30001, 30002. The mean of this data set is 30001 and the sum of squares is, once again, 2. Change the last line of each program to:

```
500 data 30000, 30001, 30002
```

and RUN each of them. Program 1 gives correct answers but Program 2 gives 1.75 for the sum of squares. Things get even worse if we try the data set 100000, 100001, 100002. The sum of the squares is still 2. Once again, Program 1 gives correct answers, but Program 2 is way off with a sum of squares of 8!

We have seen that the calculator algorithm used in Program 2 works well when the data consists of small values, but as the data values get larger, the results get progressively worse. This happens because intermediate values X2 in line 140 and T2 in line 170 are very large and cannot be represented exactly by the computer, so they are rounded off. One rounded number is subtracted from another rounded number in line 180 for the final wrong answer. You may protest that numbers like those used in the examples here are not likely to come up in ordinary problems. This may be true, but with larger data sets, i.e. many more than three data values, the same kinds of rounding errors can occur when much smaller values are processed. In other words, a tiny set of large numbers serves as a proxy for a more realistic data set.

There is a third kind of algorithm we can use that is faster than the two pass method and also very accurate. This method is called an "updating algorithm". The idea is to read through the data only once, but each time a data value is read, the mean and sum of squares of all data values up to the most recent one, are computed. There are actually several different updating algorithms. One of them is implemented in Program 3. If you try running Program 3 with each of the sample data sets given above, you will find that all the answers are correct. The updating algorithm offers both speed and accuracy and is the method used by many sophisticated scientific packages on mainframe computers.

This discussion of algorithms for computation of sums of squares is directed not only to programmers of scientific software, but to users of applications programs as well. Statistical packages use sums of squares in the calculation of variances, standard deviations, and other statistics. Some spreadsheet and database programs also calculate variances and standard deviations. Unfortunately, many programs, even relatively expensive commercial products, give wrong answers! The user must be wary. The simple data sets given above, while not infallible tests for all inaccuracies, are usually good indicators of program reliability. If the program gives the "variance" of a group of numbers, it should give a value of 1 for any of the sample data sets. The "standard deviation" is the square root of the variance, thus, in the present case, it should also be 1.*

Summary

Numerical results, such as sums of squares, can be calculated in various ways. Formulas that are mathematically equivalent may be very different with regard to speed and/or accuracy when implemented on a computer. Programmers must take care to select algorithms that do not allow intermediate computations to wander beyond the limits of a computer's precision. If proper care is taken, even inexpensive home computers can produce perfectly acceptable levels of accuracy in difficult numerical problems.

* The usual definition of variance is the sum of squares divided by one less than the sample size, e.g. in the present case, 2/2 = 1. An alternative definition uses the sample size in the denominator, or 2/3 = 0.67. In the latter example the standard deviation would be the square root of 0.67, or about 0.82.

Program 1

```
NC  100 rem  two-pass algorithm
CK  110 t=0:s=0
KD  120 rem  find mean
JD  130 for i=1 to 3
CK  140 read x
OP  150 t=t+x
PH  160 next i
NL  170 m=t/3
MM  180 restore
PO  190 rem  find sum of squares
PH  200 for i=1 to 3
IO  210 read x
OA  220 s=s+(x-m)*(x-m)
FM  230 next i
CM  240 print "mean = ",m
LF  250 print "sum sq = ",s
JF  500 data 0,1,2
```

Program 2

```
FK  100 rem  calculator algorithm
EB  110 x2=0:t=0
PC  120 for i=1 to 3
IJ  130 read x
AB  140 x2=x2+x*x
OP  150 t=t+x
PH  160 next i
IF  170 t2=t*t
JO  180 s=x2-(t2/3)
BN  190 m=t/3
KJ  200 print "mean = ",m
DD  210 print "sum sq = ",s
JF  500 data 0,1,2
```

Program 3

```
AE  100 rem  updating algorithm
AE  110 n=0: m=0: s=0
PC  120 for i=1 to 3
IJ  130 read x
CI  140 n=n+1
MD  150 s=s+(x-m)*(x-m)-(x-m)*(x-m)/n
AB  160 m=m+(x-m)/n
JI  170 next i
GI  180 print "mean = ",m
PB  190 print "sum sq = ",s
JF  500 data 0,1,2
```

The Projector

Ian Adam
Vancouver, B.C.

As close to 3D as possible short of holographics. . .

Abstract

This article builds upon the high–resolution drawing routines introduced in Volume 5, Issue 6 of The Transactor. It presents a BASIC program to construct a three–dimensional plot, using these routines. A matrix generated from either a mathematical formula or empirical data may be plotted. The plot is self scaling, and includes a title.

The Projector

In Volume 5 Issue 6 of the Transactor, Gary Kiziak introduced an excellent high–resolution graphics utility for the Commodore 64. That utility gives the programmer access to the extensive graphics that are available with the 64 but not supported by BASIC. The utility resides in the free RAM at $C000, and uses direct SYS calls to plot on the high–res screen at $E000.

Well, the article sounded good, so I typed in the utility and gave it a try. The routines work well, and are easy to access, with a good range of functions available. I will also look forward to the circle and ellipse routine that Gary promised. About the only complaint I could think of is that the hi–res screen at $E000, hidden underneath the kernal ROM, is inaccessible to my screen dump routine. Solving that may take some ingenuity.

One thing leads to another, however, and I soon found myself experimenting long into the night with the new routines. What came out of the mill was an interesting program, reproduced here, which projects a three–dimensional representation of a matrix of data. You have probably seen similar plots before, as they make for a good way to show off the capabilities of a computer's graphics, or indeed those of a printer. Thus, they are sometimes favoured by manufacturers. In addition to demonstrations, a couple of other applications spring to mind. By feeding in different mathematical functions, you could use this program to help visualize and understand the meaning of trigonometric formulae. A completely different application would be to plot up the empirical results of a scientific experiment, or ground contours, etc.

The 64 doesn't have a holographic display screen (at least not yet, anyway), so the 3–dimensional data has to be confined to a 2–dimensional display. This is achieved by viewing the Y–coordinate at an angle, theta, from the X–axis. The horizontal component of Y is expressed as a function of COS(theta), with the vertical component based on SIN(theta). The X–coordinate is simply viewed horizontally, while the dependent variable Z(X,Y) is viewed vertically. Both have to be modified by a scale factor in order to fit comfortably on the 64's screen.

The Program

The program itself is straightforward; these comments should help in understanding it, and as a guide to any modifications you may have in mind:

Lines 150–210 set up the plotting calls and load the machine language.

Lines 240 and 250 contain the values for M and N, the number of lines in the plot in the X–direction and Y–direction respectively. There is a trade–off here. . . higher values will give better resolution, but at the expense of speed. 260 and 270 set up the resulting arrays.

Lines 290 to 340 are the loop to calculate function values. This is where you can substitute other expressions in line 330, for different plots. Lines 390 to 490 contain other formulae to experiment with. If you do change the formula, remember to change the title in line 360.

Lines 530 to 580 are where the viewing angle is set, or defaulted to 60 degrees. This is a simplification, since in reality there are two angles to be set, one a rotation in reference to the X–Y axes, and the other the elevation above the X–Y plane. For simplicity, these have been combined into one composite angle. Line 580 converts to radians.

Lines 590 to 710 proceed to create a base grid with these parameters to fit the 64's screen, and set up the necessary arrays. The X–coordinate is scaled to fit across the screen. The projection of the Y–coordinate horizontally and vertically is specified in the YHRIZ and YVERT arrays. The Z–coordinate plots the array Z(X,Y) vertically.

Lines 730 to 820 calculate the vertical scale, a critical factor in the plot. Without going into all the details, the largest scale is selected to contain the plot on the screen.

Lines 840 to 900 use this scale to construct a second matrix, R(X,Y).

Lines 930 and 940 set up the high–res screen, with orange plotting on a black background.

With all that preparation out of the way, most of the rest of the program actually projects the data. Lines 960 to 1020 plot the horizontal lines, while 1040 to 1100 do the vertical lines. 1120 to 1190 draw the base of the projection, while 1220 prints the title.

Finally, line 11250 allows you to view the result, then press any key to return to the text screen and decide how to proceed.

How To Use It

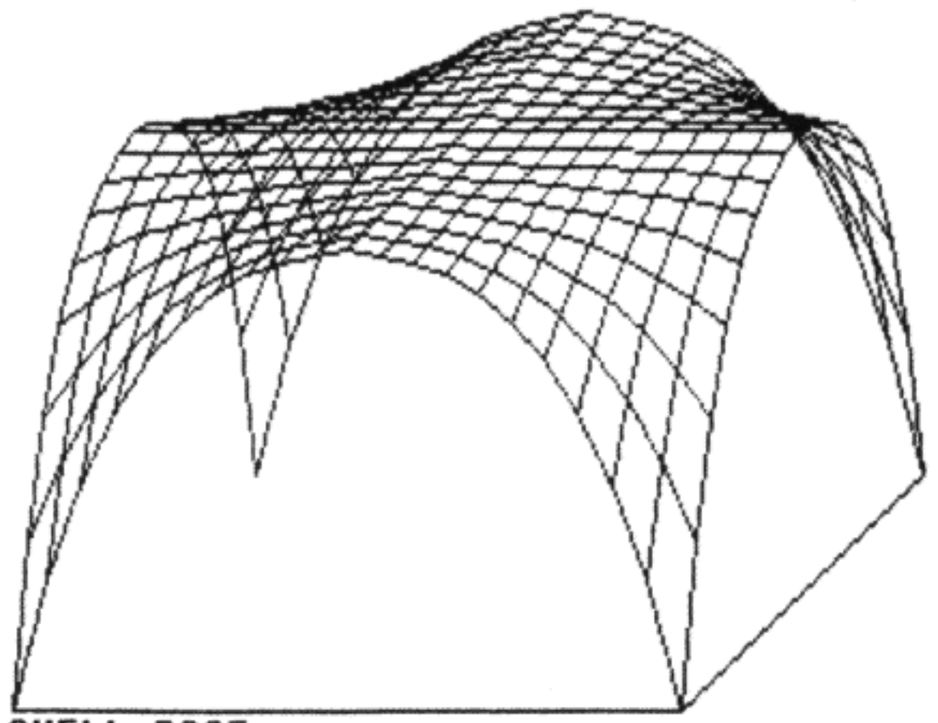
Type in the Projector program, and save it on the same disk as the Hires machine language file. Make sure you have a copy saved, then RUN. The program will LOAD the Hires file as its first step. Be patient; preparing the data for plotting will take anywhere up to a minute or so, depending on its complexity.

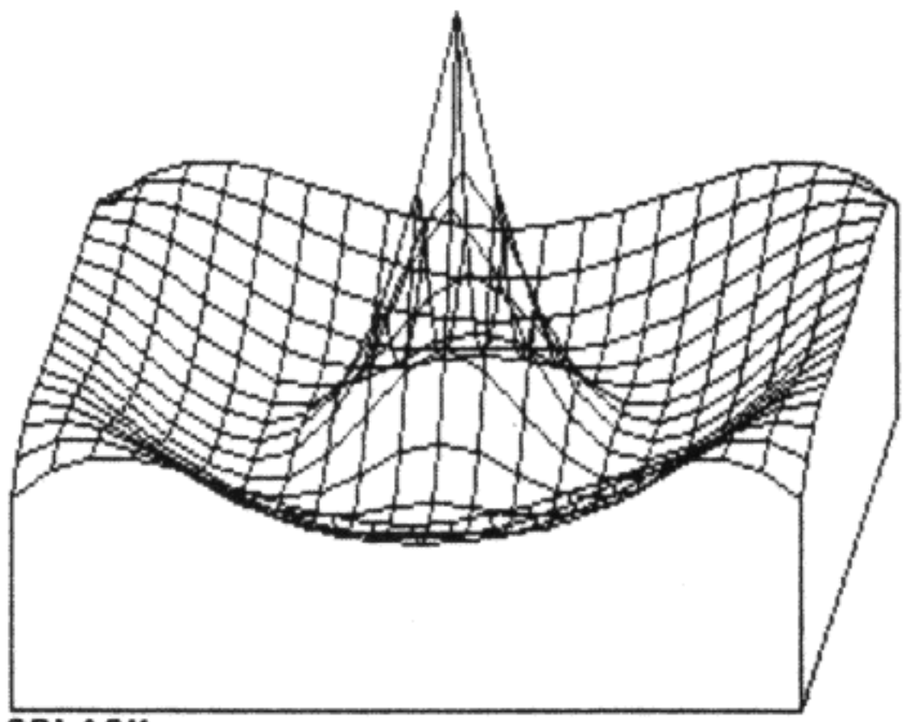
You will then be asked what viewing angle you want; enter a value from 0 to 90 degrees. A small angle will emphasize the relief of the plot, but may hide some details. A large angle gives a broader overview. If in doubt, simply press 'return' for the default value of 60 degrees.

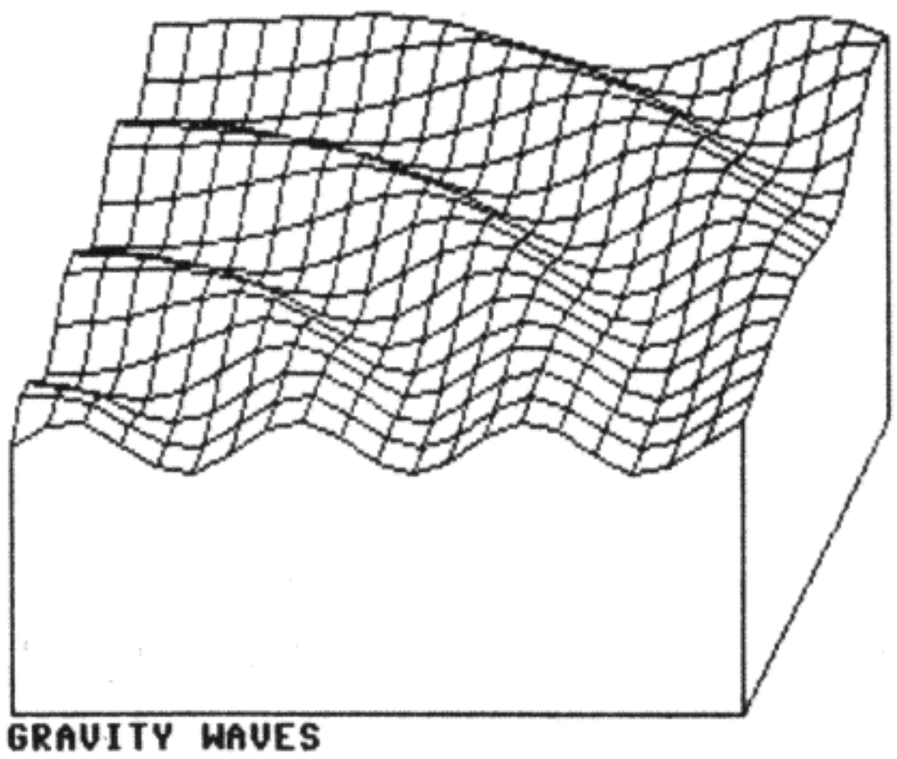
The plotting itself is interesting to watch, and only takes about 15 seconds. After viewing the plot, press any key to return to the text screen. You will then have a choice of reviewing the same data from a different angle, or ending the program. After ending, try one of the other expressions in lines 390 to 490. Simply renumber one of the formulae as line 330, and the corresponding title as 360. Remove the REM in each case, then RUN. You may wish to substitute other expressions of your own – the possibilities are limitless. If you need to plot empirical information, enter it at the end as DATA statements, then replace line 330 with READ Z(X,Y).

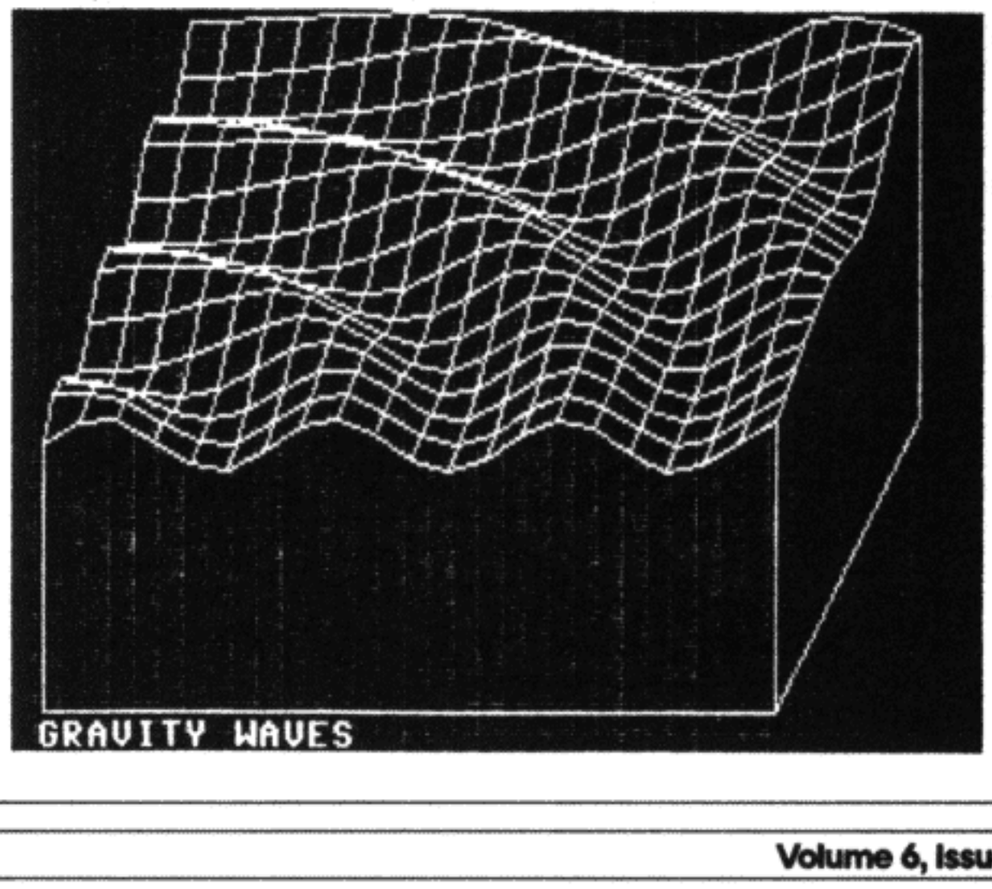
The program is fairly forgiving of errors, since it is self–scaling. It will accept and plot some small negative numbers, though anything excessive will stop it with an error message. The easiest way to correct a negative number is to add a constant to the expression, as shown in line 410.

This program grew out of Gary Kiziak's utility routines; you, in turn, may wish to embellish it further. Add refinements, parallax, new formulae, a hidden–line algorithm, whatever – there should be enough to keep you busy on a rainy day. Sometimes the best ideas don't happen all at once; like a lawn in the rain, they just grow!


SHELL ROOF


SPLASH


GRAVITY WAVES


GRAVITY WAVES

Editor's Note

PICTRANS, the subroutine below, will transfer an 8K hi-res screen at $E000 to memory at $2000. Add these lines to Ian's program and simply GOSUB 50000. Once transferred, a printout can easily be made with either PICPRINT (Volume 5, Issue 03, Disk 2) or BIGPRINT (Volume 5, Issue 06, Disk 5) or any similar program. BIGPRINT will be included on The Transactor Disk (Disk 9) for this issue.

```
NE 50000 rem* data loader for "pictrans" *
BN 50010 cs=0
LI 50020 for i=828 to 869:read a:poke i,a
JL 50030 cs=cs+a:next i
MO 50040 :
PM 50050 if cs<>6573 then print "!data error!" : end
AH 50060 sys 828
CL 50070 return
EB 50080 :
DP 50090 data 169,  32, 133, 254, 169, 224, 133, 252
HI 50100 data 169,   0, 133, 251, 133, 253, 160,   0
MM 50110 data 120, 165,   1,  72,  41, 253, 133,   1
BP 50120 data 177, 251, 145, 253, 200, 208, 249, 230
PN 50130 data 254, 230, 252, 208, 243, 104, 133,   1
KK 50140 data  88,  96
```

```
CM | 100 rem the projector - perspective plotter
HO | 110 rem by  ian adam  vancouver, bc
MN | 120 rem requires hires plotting routines
EL | 130 rem from the transactor vol 5 iss 06
AA | 140 :
PK | 150 rem setup
PN | 160 hi = 49152: co = 49173: dr = 49155
NA | 170 mo = 49161: pr = 49182: dm = 49167: te = 49179
LJ | 180 cd$ = chr$(17)
CD | 190 :
MH | 200 if peek(hi + 1) = 194 then 240
JI | 210 load " hires " ,8,1
AF | 220 :
HO | 230 rem parameters
GM | 240 m = 20: rem x-dimension
IN | 250 n = 16: rem y-dimension
AL | 260 dim z(m,n),r(m,n)
LO | 270 dim xh(m),yh(n),yv(n)
MI | 280 :
DE | 290 rem data to plot
NL | 300 print " creating data "
HG | 310 for x = 0 to m
HH | 320 for y = 0 to n
EJ | 330 z(x,y) = 12*x + 10*y-1.25*x*y
CK | 340 next y: print x: next x
CN | 350 :
OF | 360 a$ = " hyperbolic paraboloid " : rem title
GO | 370 :
CP | 380 rem insert other expressions in 330, and change title in 360
FA | 385 rem (change line # of desired function and title below)
PM | 390 rem z(x,y) = x*x-x*x*x/22 + 75*y-12*y*y  + y*y*y/2
MB | 400 rem a$ = " contours "
HP | 410 rem z(x,y) = 560-exp(sqr(abs((x-10)*(y-8)/2)))
PP | 420 rem a$ = " shell roof "
FN | 430 rem tm = sqr(x*x + 1.5*y*y): z(x,y) = 10  + sin(tm) + y/4
CF | 440 rem a$ = " gravity waves "
CL | 450 rem tm = sqr((x-10)↑2 + (y-8)↑2): z(x,y) = 150-tm*55
   |       + tm*tm*8-tm*tm*tm/3
AA | 460 rem a$ = " splash "
CM | 470 rem a = 20-abs(x-10): b = 18-abs(y-8): z(x,y) = a
   |       : ifb>athenz(x,y) = b
ID | 480 rem a$ = " house "
KA | 490 rem z(x,y) = y + (8-y)*((x>4)and(x<16))*((y>3)and(y<13))
LF | 500 rem a$ = " plateau "
FL | 510 rem or read empirical results from data
MH | 520 :
BD | 530 rem projection
KF | 540 theta = 60: rem default angle
GN | 550 print cd$ " enter viewing angle, or press return
AN | 560 print  " for 60 degrees:
AG | 570 input th
HH | 580 th = th*3.14159265/180
BB | 590 tmp = 120*cos(th)
OH | 600 xgrid = int((309-tm)/m)
EB | 610 ygrid = int(96*sin(th)/n)
AD | 620 ystp = int(tm/n)
KO | 630 :
JD | 640 rem calculate offsets
LL | 650 for x = 0 to m
OB | 660 xhriz(x) = 10 + x*xg
CK | 670 next
PN | 680 for y = 0 to n
EK | 690 yhriz(y) = y*ys
FE | 700 yvert(y) = 10 + y*yg
KM | 710 next
EE | 720 :
AA | 730 rem vertical scaling
HB | 740 print " scaling data
KF | 750 vscalar = 9e9
PC | 760 for y = 0 to n
FM | 770 a = 0: for x = 0 to m
CP | 780 if z(x,y)>a then a = z(x,y)
CL | 790 next: rem find highest point on line
OG | 800 if a then tmp = (199-yv(y))/a
JB | 810 if  vs>tm  then  vs = tm
KO | 820 next: rem select best feasible scale
CL | 830 :
CP | 840 rem calculate rise
CJ | 850 print " . . .still scaling!
DJ | 860 for y = 0 to n
HM | 870 tm = yv(y)
BK | 880 for x = 0 to m
HF | 890 r(x,y) = z(x,y)*vs + tm
JJ | 900 next x,y
CA | 910 :
FB | 920 rem set up screen
GE | 930 sys hi,0,0,8
GH | 940 sys dm,1
KC | 950 :
PE | 960 rem plot horizontal lines
BA | 970 for y = 0 to n
JB | 980 tm = yh(y)
GJ | 990 sys mo,tm + 10,r(0,y)
KB | 1000 for x = 1 to m
LN | 1010 sys dr,tm + xh(x),r(x,y)
BB | 1020 next x,y
KH | 1030 :
OO | 1040 rem plot vertical lines
LE | 1050 for x = 0 to m
EG | 1060 tm = xh(x)
EN | 1070 sys mo,tm,r(x,0)
AH | 1080 for y = 1 to n
AD | 1090 sys dr,tm + yh(y),r(x,y)
DG | 1100 next y,x
KM | 1110 :
LN | 1120 rem draw box
LP | 1130 sys mo,10,r(0,0)
DC | 1140 sys dr,10,10
JD | 1150 sys dr,xh(m),10
GM | 1160 sys dr,xh(m),r(m,0)
MF | 1170 sys mo,xh(m),10
HI | 1180 sys dr,xh(m) + yh(n),yv(n)
AD | 1190 sys dr,xh(m) + yh(n),r(m,n)
EC | 1200 :
DL | 1210 rem title
BA | 1220 sys co,13: sys pr,1,24,a$
CE | 1230 :
AP | 1240 rem wait for human
JB | 1250 wait 198,1: get b$
ID | 1260 sys te
KG | 1270 :
LK | 1280 print cd$ " press r to review from another angle
FC | 1290 print  " press any other key to end
LE | 1300 wait 198,1: get b$
LG | 1310 if b$ = " r "  then 540
IC | 1320 end
```

HIRES Create

Gary's hi-res utility will be included on Disk 9 for this issue. For a complete description of the commands you'll need the first Programming Aids and Utilities issue (Volume 5, Issue 06).

```
FN 1000 rem ** hires routine - written by gary kiziak
JP 1010 rem ** creates load/run program on diskette **
AH 1020 :
HK 1030 open 15,8,15: open 8,8,8, " 0:hires,p,w "
NB 1040 input#15,e,e$,b,c: if e then close 15: print e,e$,b,c
        : stop
JA 1050 for j = 49152 to 51233: read x: print#8,chr$(x);
        : ch = ch + x: next: close8
HA 1060 if ch<>245919 then print " checksum error " : end
GI 1070 print " ** program complete ** " : end
MK 1080 :
BL 1090 data  76, 206, 197,  76, 199, 199,  76,   4
IB 1100 data 200,   0,   0,   0,   0,   0,   0,   0
ED 1110 data   0,   0,   0,   0,   0, 255, 128,   0
FF 1120 data   7, 248,   0,   0,   0,   0,   0,   0
MJ 1130 data   0,   0,   0,   1,   0,  15, 240, 240
KF 1140 data   0,   0, 208,   0,   0,   0,   0, 173
EN 1150 data  58, 192, 208,  27, 173,   0, 221, 141
FD 1160 data  57, 192, 173,  24, 208, 141,  58, 192
HE 1170 data 173,  17, 208, 141,  59, 192, 173,  22
AB 1180 data 208, 141,  60, 192,  32, 110, 192,  96
OK 1190 data 173,   0,   3, 201, 231, 208,   7, 173
JE 1200 data   1,   3, 201, 192, 240,  44, 173,   0
IJ 1210 data   3, 141, 234, 192, 173,   1,   3, 141
MP 1220 data 235, 192, 169, 231, 141,   0,   3, 169
OO 1230 data 192, 141,   1,   3, 173,   2,   3, 141
IC 1240 data  41, 193, 173,   3,   3, 141,  42, 193
LP 1250 data 169,   8, 141,   2,   3, 169, 193, 141
HJ 1260 data   3,   3,  96, 173,  58, 192, 240,  26
NC 1270 data 141,  24, 208, 173,  57, 192, 141,   0
OF 1280 data 221, 173,  59, 192, 141,  17, 208, 173
FD 1290 data  60, 192, 141,  22, 208, 169,   0, 141
GF 1300 data  58, 192,  96,  72, 169, 127, 141,  13
BC 1310 data 220, 165,   1, 141,  56, 192,  41, 253
JD 1320 data 133,   1, 104,  96,  72, 173,  56, 192
OH 1330 data 133,   1, 169, 129, 141,  13, 220, 104
DN 1340 data  96,  16,   3,  76, 139, 227, 142,  13
LC 1350 data   3,  44,  76, 192,  16, 245, 169,   0
LH 1360 data 133,  20, 169,   0, 133,  21, 162, 250
CE 1370 data 154, 169, 167,  72, 169, 233,  72,  76
KM 1380 data 163, 168,  32, 169, 192, 173, 234, 192
EJ 1390 data 141,   0,   3, 173, 235, 192, 141,   1
NM 1400 data   3, 173,  41, 193, 141,   2,   3, 173
PD 1410 data  42, 193, 141,   3,   3, 169,   0, 141
KL 1420 data  76, 192,  76, 131, 164, 164, 254, 240
LO 1430 data  13, 160,   0, 145, 251, 200, 208, 251
MO 1440 data 230, 252, 198, 254, 208, 243, 164, 253
CC 1450 data 240,  10, 136, 240,   5, 145, 251, 136
PN 1460 data 208, 251, 145, 251,  96,  32, 201, 192
NH 1470 data 160,   0, 132, 251, 160, 204, 132, 252
MN 1480 data 160, 232, 132, 253, 160,   3, 132, 254
DA 1490 data  32,  43, 193, 169,   0, 133, 251, 169
KB 1500 data 224, 133, 252, 169,  64, 133, 253, 169
CC 1510 data  31, 133, 254, 169,   0,  32,  43, 193
BK 1520 data  76, 218, 192,  32, 253, 174,  32, 138
OL 1530 data 173,  32, 247, 183, 166,  21, 165,  20
DH 1540 data  96,  32, 253, 174,  32, 124, 193, 141
CJ 1550 data  43, 192, 142,  44, 192,  32, 121, 193
AN 1560 data 141,  45, 192, 142,  46, 192, 169,  63
PM 1570 data 162,   1,  44,  53, 192,  16,   4, 169
HJ 1580 data 159, 162,   0, 205,  43, 192, 138, 237
IM 1590 data  44, 192, 176,   3,  76,  72, 178, 169
JL 1600 data 199, 205,  45, 192, 169,   0, 237,  46
GJ 1610 data 192, 144, 241,  96,  32,  77, 192,  32
AN 1620 data 121, 193, 240,   2, 169, 128, 141,  53
FN 1630 data 192,  32, 121,   0, 240,   3,  32,  30
LE 1640 data 194, 173,   0, 221,   9,   3,  73,   3
NG 1650 data 141,   0, 221, 173,  24, 208,  41,   7
KK 1660 data   9,   8,   9,  48, 141,  24, 208, 173
OP 1670 data  17, 208,   9,  32, 141,  17, 208,  44
DM 1680 data  53, 192,  16,  12, 173,  22, 208,   9
FH 1690 data  16, 141,  22, 208, 169,   3, 208,  10
KC 1700 data 173,  22, 208,  41, 239, 141,  22, 208
JN 1710 data 169,   7, 141,  54, 192,  73, 255, 141
OJ 1720 data  55, 192, 169, 255, 141,  51, 192,  96
BO 1730 data 169,   1, 141,  65, 192, 173,  67, 192
EF 1740 data 141,  66, 192, 169, 128, 141,  52, 192
BB 1750 data  32, 135, 193, 173,  45, 192,  10,  10
BN 1760 data  10,  10, 141,  62, 192, 141,  70, 192
BD 1770 data 173,  43, 192,  41,  15, 141,  61, 192
KB 1780 data  44,  53, 192,  48,  12,  13,  62, 192
EE 1790 data 141,  62, 192, 141,  70, 192,  76,  75
DJ 1800 data 193, 141,  33, 208,  32, 121, 193,  41
PG 1810 data  15, 141,  63, 192,  32, 121, 193, 141
FJ 1820 data  64, 192, 173,  62, 192,  76,  75, 193
BJ 1830 data  32, 135, 193, 162,   3, 189,  43, 192
EJ 1840 data 157,  39, 192, 202,  16, 247,  96,  56
JK 1850 data 169, 199, 237,  41, 192,  72,  74,  74
LH 1860 data  74, 133, 252, 160,   0, 132, 251,  74
OD 1870 data 102, 251,  74, 102, 251, 101, 252, 133
IC 1880 data 252, 173,  39, 192, 174,  40, 192,  45
FP 1890 data  55, 192,  44,  53, 192,  16,   6,  10
DN 1900 data  72, 138,  42, 170, 104,  24, 101, 251
NH 1910 data 133, 251, 138, 101, 252, 133, 252, 104
GJ 1920 data  41,   7,  24, 101, 251, 133, 251, 133
BO 1930 data 253, 144,   2, 230, 252, 165, 252,  74
EL 1940 data 102, 253,  74, 102, 253,  74, 102, 253
IJ 1950 data 133, 254,  44,  53, 192,  48,  16,  24
OI 1960 data 169,   0, 101, 253, 133, 253, 169, 204
PP 1970 data 101, 254, 133, 254,  76, 249, 194, 173
GP 1980 data  65, 192, 201,   3, 144, 234,  24, 169
MI 1990 data   0, 101, 253, 133, 253, 169, 216, 101
HI 2000 data 254, 133, 254,  24, 165, 251, 105,   0
JP 2010 data 133, 251, 165, 252, 105, 224, 133, 252
EE 2020 data 173,  39, 192,  45,  54, 192, 170,  96
DK 2030 data 169,   0, 168,  44,  52, 192,  16,   4
MN 2040 data 112,  20,  80,  15,  36,   2,  48,   9
HC 2050 data 169, 255, 133,   2,  36, 107,  48,   1
NE 2060 data  96, 177, 251,  77,  51, 192,  44,  53
GG 2070 data 192,  48,  10,  61,  86, 195, 133,  97
AM 2080 data 189,  86, 195, 208,   8,  61,  94, 195
MK 2090 data 133,  97, 189,  94, 195,  73, 255,  49
JE 2100 data 251,   5,  97, 145, 251, 177, 253,  45
NH 2110 data  66, 192,  13,  70, 192, 145, 253,  96
CP 2120 data 128,  64,  32,  16,   8,   4,   2,   1
HK 2130 data 192,  48,  12,   3,  32, 110, 194,  32
BC 2140 data 121,   0, 240,  11,  32, 228, 196,  32
CI 2150 data 121,   0, 240,   3,  32, 214, 196,  32
EB 2160 data 201, 192,  32, 125, 194,  32,  14, 195
FG 2170 data  76, 218, 192, 169,   1, 149, 106, 169
EL 2180 data   0, 149, 107,  56, 189,  43, 192, 253
KH 2190 data  39, 192, 149,  98, 189,  44, 192, 253
OA 2200 data  40, 192, 149,  99,  16,  20, 169, 255
EF 2210 data 149, 106, 149, 107,  56, 169,   0, 245
BF 2220 data  98, 149,  98, 169,   0, 245,  99, 149
AC 2230 data  99,  96,  21,  98, 208,   4, 149, 106
KH 2240 data 149, 107,  96, 165,  99,  74, 133, 103
GC 2250 data 165,  98, 106, 133, 102,  24, 169,   0
GG 2260 data 229,  98, 133, 104, 169,   0, 229,  99
FM 2270 data 133, 105,  96,  24, 165, 102, 101, 100
MM 2280 data 133, 102, 170, 165, 103, 101, 101, 133
KF 2290 data 103, 197,  99, 144,  19, 208,   4, 228
IG 2300 data  98, 144,  13, 138,  56, 229,  98, 133
```

```
CF 2310 data 102, 165, 103, 229,  99, 133, 103,  56
GO 2320 data  96,  32, 135, 193,  32, 121,   0, 240
PK 2330 data  44, 201, 164, 208,  16,  32, 113, 194
AC 2340 data  32, 115,   0,  32, 138, 193,  32, 121
MB 2350 data   0, 201,  44, 208,  13,  32, 228, 196
HJ 2360 data  32, 121,   0, 201,  44, 208,   3,  32
NB 2370 data 214, 196,  32,  43, 196,  32, 121,   0
FG 2380 data 201, 164, 240, 220,  96,  32, 201, 192
MG 2390 data 162,   0, 134,   2,  32, 129, 195, 162
DI 2400 data   2,  32, 129, 195, 165,  98, 197, 100
MD 2410 data 165,  99, 229, 101, 144,  62,  32, 185
BM 2420 data 195,  36, 107,  16,  10,  32, 113, 194
OL 2430 data  56, 169,   0, 229, 108, 133, 108,  32
FJ 2440 data 125, 194,  32,  14, 195, 230, 104, 208
HM 2450 data   4, 230, 105, 240, 102, 238,  39, 192
HK 2460 data 208,   3, 238,  40, 192,  32, 209, 195
KL 2470 data 144,   9,  24, 173,  41, 192, 101, 108
BM 2480 data 141,  41, 192,  32, 125, 194,  32,  14
KA 2490 data 195,  76,  91, 196, 162,   1, 181,  98
FA 2500 data 180, 100, 149, 100, 148,  98, 202,  16
OB 2510 data 245,  32, 185, 195,  36, 107,  16,  10
HB 2520 data  32, 113, 194,  56, 169,   0, 229, 108
DH 2530 data 133, 108,  32, 125, 194,  32,  14, 195
FE 2540 data 230, 104, 240,  31,  24, 173,  41, 192
PA 2550 data 101, 108, 141,  41, 192,  32, 209, 195
MA 2560 data 144,   8, 238,  39, 192, 208,   3, 238
PC 2570 data  40, 192,  32, 125, 194,  32,  14, 195
PI 2580 data  76, 166, 196,  36, 107,  16,   3,  32
MK 2590 data  14, 195,  32, 113, 194,  76, 218, 192
FM 2600 data  32, 121, 193,  41,   3,  73,   3, 106
BL 2610 data 106, 106, 141,  52, 192,  96,  32, 121
FH 2620 data 193,  41,   3, 240,  27,  44,  53, 192
ME 2630 data  16,  22, 141,  65, 192, 170, 189,  61
CM 2640 data 192, 141,  70, 192, 189,  66, 192, 141
GO 2650 data  66, 192, 189,   7, 197, 141,  51, 192
GK 2660 data  96,   0,  85, 170, 255,  32, 121, 193
NH 2670 data  10,  10,  10,  10, 141,  62, 192,  44
GL 2680 data  53, 192,  48,   9,  13,  61, 192, 141
NM 2690 data  62, 192,  76,  51, 197,  32, 121, 193
EJ 2700 data  41,  15, 141,  63, 192,  32, 121, 193
BL 2710 data  41,  15, 141,  64, 192, 174,  65, 192
AG 2720 data 189,  61, 192, 141,  70, 192, 189,  66
BG 2730 data 192, 141,  66, 192,  96,  32, 110, 194
PB 2740 data  32, 135, 193, 162,   3, 189,  43, 192
NF 2750 data 157,  47, 192, 202,  16, 247,  32, 121
BJ 2760 data   0, 240,  11,  32, 228, 196,  32, 121
GP 2770 data   0, 240,   3,  32, 214, 196,  24, 173
PI 2780 data  39, 192, 109,  47, 192, 141,  43, 192
CJ 2790 data 173,  40, 192, 109,  48, 192, 141,  44
EE 2800 data 192, 173,  41, 192, 141,  45, 192, 173
JI 2810 data  42, 192, 141,  46, 192,  32, 156, 193
OD 2820 data  32,  43, 196,  56, 173,  45, 192, 237
ML 2830 data  49, 192, 141,  45, 192, 173,  46, 192
FK 2840 data 237,  50, 192, 141,  46, 192,  32, 181
LG 2850 data 193,  32,  43, 196,  56, 173,  43, 192
EN 2860 data 237,  47, 192, 141,  43, 192, 173,  44
JP 2870 data 192, 237,  48, 192, 141,  44, 192,  32
NH 2880 data  43, 196,  24, 173,  45, 192, 109,  49
IL 2890 data 192, 141,  45, 192, 173,  46, 192, 109
GK 2900 data  50, 192, 141,  46, 192,  76,  43, 196
KJ 2910 data 169,   0, 133, 251, 133, 252,  32, 241
MK 2920 data 183, 224,  40, 144,   3,  76,  72, 178
EP 2930 data 142,  73, 192,  32, 241, 183, 142,  74
GN 2940 data 192, 138, 240,  18, 224,  25, 176, 237
DK 2950 data  24, 165, 251, 105,  40, 133, 251, 144
HL 2960 data   2, 230, 252, 202, 208, 242,  24, 173
LL 2970 data  73, 192, 101, 251, 133, 251, 133, 253
DL 2980 data 133,   3, 169,   0, 101, 252, 133, 252
FJ 2990 data  24,  72, 105, 216, 133, 254, 104, 105
PF 3000 data 204, 133,   4,   6, 251,  38, 252,   6
AO 3010 data 251,  38, 252,   6, 251,  38, 252,  24
CM 3020 data 165, 252, 105, 224, 133, 252,  32, 253
NH 3030 data 174,  32, 158, 173,  32, 143, 173,  32
GP 3040 data 166, 182, 170, 160,   0, 232, 202, 208
CA 3050 data   1,  96, 177,  34,  32,  73, 198, 200
IH 3060 data  76,  60, 198, 133, 215, 138,  72, 152
GE 3070 data  72, 165, 215,  48,  17, 201,  32, 144
LB 3080 data  28, 201,  96, 144,   4,  41, 223, 208
HG 3090 data   2,  41,  63,  76, 110, 199,  41, 127
AB 3100 data 201, 127, 208,   2, 169,  94, 201,  32
DJ 3110 data 144, 125,  76, 108, 199, 201,  14, 208
BH 3120 data   6,  32, 219, 199,  76, 160, 199, 201
HF 3130 data  17, 208,  11, 162,  40,  32,  71, 199
GK 3140 data 202, 208, 250,  76, 160, 199, 201,  18
FL 3150 data 208,   8, 169,   1, 141,  75, 192,  76
NI 3160 data 160, 199, 201,  29, 208,   6,  32,  71
BA 3170 data 199,  76, 160, 199, 162,   3,  44, 162
CL 3180 data  15, 221, 205, 198, 240,   6, 202,  16
HD 3190 data 248,  76, 160, 199, 189, 221, 198,  10
IJ 3200 data  10,  10,  10, 141,  62, 192,  44,  53
LP 3210 data 192,  48,   6,  13,  61, 192, 141,  62
JJ 3220 data 192,  32,  51, 197,  76, 160, 199,   5
MH 3230 data  28,  30,  31,  16,  28,  30,  31,   1
CI 3240 data  21,  22,  23,  24,  25,  26,  27,   1
BM 3250 data   2,   5,   6,   0,   4,   7,   3,   8
GK 3260 data   9,  10,  11,  12,  13,  14,  15, 201
OP 3270 data  14, 208,   6,  32, 216, 199,  76, 160
GP 3280 data 199, 201,  17, 208,  11, 162,  40,  32
BG 3290 data  28, 199, 202, 208, 250,  76, 160, 199
DD 3300 data 201,  18, 208,   8, 169,   0, 141,  75
KG 3310 data 192,  76, 160, 199, 201,  29, 208, 143
II 3320 data  32,  28, 199,  76, 160, 199, 165, 253
GB 3330 data 208,   2, 198, 254, 198, 253, 165,   3
OJ 3340 data 208,   2, 198,   4, 198,   3,  56, 165
CB 3350 data 251, 233,   8, 133, 251, 165, 252, 233
AP 3360 data   0, 133, 252, 165, 251, 201,   0, 165
BH 3370 data 252, 233, 224, 176,   3,  32,  71, 199
IN 3380 data  96, 230, 253, 208,   2, 230, 254, 230
FC 3390 data   3, 208,   2, 230,   4,  24, 169,   8
JD 3400 data 101, 251, 133, 251, 144,   2, 230, 252
HI 3410 data 165, 251, 201,  64, 165, 252, 233, 255
GM 3420 data 144,   3,  32,  28, 199,  96,   9,  64
BP 3430 data 174,  75, 192, 240,   2,   9, 128,  32
EB 3440 data 165, 199, 160,   7,  32, 230, 199, 177
DI 3450 data   5, 145, 251, 136,  16, 249,  32, 245
BD 3460 data 199, 200, 173,  61, 192,  44,  53, 192
AP 3470 data  16,   8, 173,  64, 192, 145, 253, 173
BM 3480 data  63, 192,  13,  62, 192, 145,   3,  32
NC 3490 data  71, 199, 104, 168, 104, 170,  96, 133
ND 3500 data   5, 169,   0, 133,   6,   6,   5,  38
EB 3510 data   6,   6,   5,  38,   6,   6,   5,  38
MN 3520 data   6,  24, 173,  71, 192, 101,   5, 133
IE 3530 data   5, 173,  72, 192, 101,   6, 133,   6
OO 3540 data  96,  32, 253, 174,  32, 138, 173,  32
MC 3550 data 247, 183, 166,  21, 208,   9, 165,  20
BD 3560 data 208,   3, 162, 208,  44, 162, 216, 142
HD 3570 data  72, 192, 162,   0, 142,  71, 192,  96
HF 3580 data 173,  14, 220,  41, 254, 141,  14, 220
FB 3590 data 165,   1,  41, 251, 133,   1,  96, 165
CO 3600 data   1,   9,   4, 133,   1, 173,  14, 220
MJ 3610 data   9,   1, 141,  14, 220,  96,  32, 121
CO 3620 data   0, 240,  15,  32, 110, 192,  32, 121
PJ 3630 data 193, 141, 245, 192, 142, 249, 192, 169
FI 3640 data 128,  44, 169,   0, 141,  76, 192,  96
IL 3650 data   0,   0
```

Microsecond Timer For The Commodore 64

Zoltan Szepesi
Pittsburgh, PA

Count microseconds for up to 70 minutes

I. Introduction.

The Commodore 64 computer provides four different ways for the measurement of elapsed time intervals. These methods can be used to measure time intervals of limited values and of limited precision. The first three methods are useful only for longer time intervals, up to 12 hours. However, the fourth method, which is the subject of this paper, enables one to measure down to 1 microsecond (one millionth of a second) and up to 70 minutes. The following are the four methods in question:

1. The TI$ software clock, which gives the time in Hours, Minutes and Seconds (as HHMMSS). It is useful when a precision of + or – one second is satisfactory.

2. The TI jiffy clock with a precision of + or – 1/60 second (16.67 milliseconds).

3. The C–64 has 2 Complex Interface Adapters (CIA#1 and CIA#2, type 6526). Both have a time of day (TOD) clock with a one tenth of a second counter, whence its precision is + or – 100ms (ms = millisecond). This clock keeps the time more accurately than the software clock (TI$ or TI), which is disturbed when I/O operations disrupt the interrupt routine or when the IRQ vector is changed.

4. Both CIA–s have two general purpose timers (A and B) with an F = 1.022730 Mh (Megahertz = 1 million Hertz) clock (in Europe the frequency is F = 0.985250 Mh, tuned to the PAL TV system). Hence, we have (in the USA) a T = 0.9777751704 microsecond (millionth of a second) counter. Both timers have 2 bytes, therefore the total time interval could be T*256*256 = 64.079 ms on one timer. However, Timer B can be linked to Timer A, so that it counts the number of times Timer A goes to zero. This way we have a 4 byte (32 bits) timer, that can count from 1 microsecond up to 70 minutes.

II. The ML Program.

Listing 1. is the source listing of the ML program. It starts at address $9F00 and takes 65+4 bytes. (You could choose another location for this program, e.g. the cassette I/O buffer, or some part of the $C000 to $CFFF address, if it is not used for other ML routine). The program is composed of two parts. The first part (START) controls the starting of the clocks, the second part (STOP) stops the clocks and stores their values in the 4 bytes following the ML program. We use the CIA#2 timers because they are more independent from the general operating system than the clocks of the CIA#1.

Timer A can be found at addresses $DD04–$DD05 (56580–56581 decimal), low byte first, high byte second. Timer B is at $DD06–$DD07. These timers count down from $FF (255) to 0. Therefore, we first store the value #$FF into the 4 bytes of the 2 timers (at addresses $9F00 to $9F0D). For starting Timer A, we have to store a value of #1 in location $DD0E, the Control Register A (TCRA). Timer B will start by storing #1 in location $DD0F, Control Register B (TCRB). Since we want to link Timer B to Timer A, to ensure that Timer B only counts when Timer A counts down to 0, we should set bit 6 of TCRB to 1, and its bit 5 to 0. This results in $40. Adding the starting byte 1 to this gives $41, of which is stored in location $DD0F.

By setting location $DD0F before $DD0E, we set the start of the counting to the end of the START routine. If we had started Timer A first, the counting would have started 2+4 cycles before the end of the START program, causing a 5.87 microsecond addition to the time measured. The RTS at the end returns the program back to Basic, where our Basic program can SYS up the the STOP routine.

The STOP routine starts at address $9F20 (7 bytes are left free for possible ML commands). Storing #0 into the control registers stops the counting. Here Timer A get stopped first. As you see, we have 6 cycles until the 0 is stored at TCRA. This takes 5.87 microseconds. After storing #0 at TCRB too, the program saves the 4 bytes of the timers at 4 locations after the end of the STOP routine (at $9F41–$9F44), from where this data can be read out, and the execution time can be calculated in Basic.

III. The BASIC Program.

Listing 2 is a Basic program/Data loader for starting the timer with a SYS command (end of line 150) for the START routine address, followed by stopping it by a SYS command (start of line 170) again for the STOP routine address. After, the program reads out the clock data and calculates the counting time in microseconds (lines 190 to 230). The following statements (240 to 280) calculate how many minutes, seconds, milliseconds, and microseconds are in this time, and print out the ones with an integer value greater than zero.

At the start of the program we also display the time measured by the TI$ software clock. This could serve to see that our program is correct, when the time interval measured is higher than several seconds.

If you want to time a Basic command or program, write it into lines 160 to 169 (renumber accordingly). Or, if it is a longer program, you can place it in a subroutine (lines 440 up – above the data statements), and enter:

```
160 GOSUB 440
```

At the end of the subroutine, use a RETURN. (First entering: 440 RETURN, you can measure how much time it takes the GOSUB+RETURN+SYS commands to be executed – about 8ms), and you can subtract this time from the total measured time).

IV. Measurement Data.

I want to show a few simple examples for the use of this program, and give some measured data.

We can measure the execution time of a simple Basic Loop program as in line 160 of Listing 2. The number of loops (N) is defined in line 140. If we first delete line 160, we will measure the time it takes to execute the SYS command of statement 170, which is about 6 ms. This value should be subtracted from the measured time of the examined Basic program. Measuring the loop of line 160, we get about T=10.6–6 = 4.6 ms with N=1, T=114 ms for N=100 and T=113.705 s for N=100000. We can also see that the execution time is 10 to 30% longer when we write out the variable I after NEXT. Furthermore, we see that when we repeat the program, the measured numbers are not exactly the same; they can differ by several percent (also, they are different in the various C–64's, depending on the ROM revision it comes with).

In the ML program we can put the STOP routine immediately after the START routine at address $9F18 and expect to measure the time for 6 cycles (LDA #0: 2 cycles and STA TCRA: 4 cycles). The program gives 5.87 microseconds, which is exactly 6*0.97777517.

If we write JMP STOP (4C 20 9F) at address $9F18, we measure 8.8 microseconds. This is just 3 cycles more than the previous time and truly the JMP command needs 3 cycles.

Instead of JMP, put JSR STOP (20 20 9F) at address $9F18, and you will measure 11.73 microseconds, what it takes to execute 12 cycles. And JSR needs 6 cycles, 3 more than the JMP command.

For a longer ML program we only need to write in:

```
JSR START (20 00 9F)
```

where we want to start the timing and:

```
JSR STOP (20 20 9F)
```

where we want to stop the timing. After the program ends we can read out the execution time with the Basic program by entering:

```
RUN 170
```

V. Conclusion.

As we see from the above data, this program measures very precisely the execution time, and can be used easily in Basic and ML programs as well.

Listing 1: ML Source Code

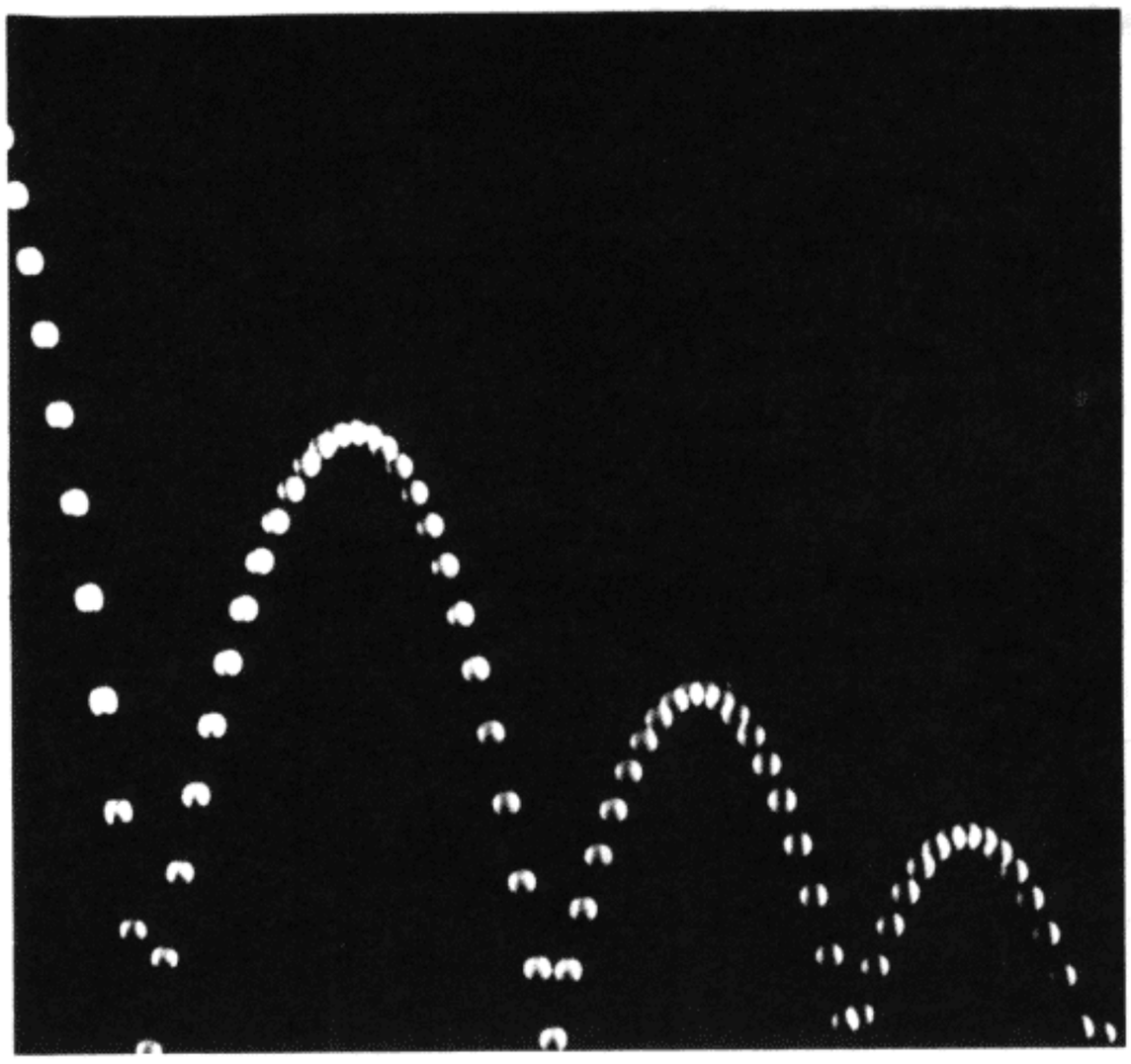
```
EL  100 rem save "0:timer64.pal",8
PI  110 rem time measurements from
LE  120 rem 1 microsecond to 70 minutes
NH  130 rem by zoltan szepesi
EH  140 rem 2611 saybrook drive
MA  150 rem pittsburgh,pa 15235
EB  160 :
BC  170 open 4,8,1,"0:timer64.obj"
PE  180 sys(700)
FN  190 .opt o4
IH  200 *=$9f00
IE  210 ;
HI  220 talo    =   $dd04   ; timer a
MJ  230 tblo    =   $dd06   ; timer b
MK  240 tcra    =   $dd0e   ; control register a
LL  250 tcrb    =   $dd0f   ; control register b
KH  260 time1   =   *+$41
KI  270 time2   =   *+$42
KJ  280 time3   =   *+$43
KK  290 time4   =   *+$44
CK  300 ;
MJ  310 start   =   *
NP  320         lda #$ff
GO  330         sta talo
MP  340         sta talo+1
MP  350         sta tblo
CB  360         sta tblo+1
FK  370         lda #$41
KP  380         sta tcrb
DH  390         lda #1
KA  400         sta tcra
GI  410         rts
KB  420 ;
BD  430 .byt $ea, $ea, $ea
CI  440 .byt $ea, $ea, $ea, $ea
ID  450 ;
CL  460 stop    =   *
BM  470         lda #0
KF  480         sta tcra
IG  490         sta tcrb
CF  500         lda talo
GL  510         sta time1
CH  520         lda talo+1
LM  530         sta time2
MH  540         lda tblo
AO  550         sta time3
MJ  560         lda tblo+1
FP  570         sta time4
AD  580         rts
EM  590 ;
ED  600 .end
```

Listing 2: Basic Demo Plus Data Loader

```
EL  100 rem save "0:timer64.bas",8
NL  110 rem microsecond to 70 minute timer by
        z.szepesi (c) 1985.
KG  120 poke 55,255: poke 56,158: clr: rem set top
        of basic below ml.
CD  130 gosub 310: rem move code into position
MH  140 n=1: rem adjust this value for demo
EG  150 print "timing started at "ti$" (hhmmss)"
        : sys40704
CB  160 for i=1 to n: next
IB  170 sys40736: print "timing finished at
        "ti$" (hhmmss)"
BO  180 a1=.97777517: a2=256*a1: a3=256*a2
        : a4=256*a3
PA  190 t1=(255-peek(40769))*a1
MP  200 t2=(255-peek(40770))*a2
OA  210 t3=(255-peek(40771))*a3
AC  220 t4=(255-peek(40772))*a4
EA  230 t=t1+t2+t3+t4: print chr$(17)"**execution
        time="chr$(145)
IE  240 m1=t/(6e+7): i1=int(m1): if i1>0 then
        print spc(17)i1" minute"
AL  250 m2=(m1-i1)*60: i2=int(m2): if i2>0 then
        print spc(17)i2" second"
AP  260 m3=(m2-i2)*1000: i3=int(m3): if i3>0 then
        print spc(17)i3" millisecond"
OK  270 m4=(m3-i3)*1000: i4=int(m4): if i4>0 then
        print spc(17) int(m4*100+.5)/100;
LG  280 print " microsecond": end
GJ  290 :
GH  300 rem ** timer64 code at $9f00 **
IL  310 for j=40704 to 40768: read x: poke j,x
        : ch=ch+x: next
JG  320 if ch<>8867 then print "checksum error": end
GG  330 return
IM  340 :
AG  350 data 169, 255, 141,   4, 221, 141,   5, 221
JE  360 data 141,   6, 221, 141,   7, 221, 169,  65
IJ  370 data 141,  15, 221, 169,   1, 141,  14, 221
EN  380 data  96, 234, 234, 234, 234, 234, 234, 234
PF  390 data 169,   0, 141,  14, 221, 141,  15, 221
NF  400 data 173,   4, 221, 141,  65, 159, 173,   5
DF  410 data 221, 141,  66, 159, 173,   6, 221, 141
MB  420 data  67, 159, 173,   7, 221, 141,  68, 159
HD  430 data  96
```

Projectile Motion

Karl J. Hildon and Chris Zamara



The following was originally presented in Transactor Volume 5, Issue 01, back before our newstand distribution days. It makes several references to "80 columns" but is readily portable to 40 columns, the 20, and the 64.

Once you understand the techniques of putting objects on your screen, you'll want to get them moving. After all, what good is a sprite if it doesn't do anything. In this article we'll discuss some simple motion techniques using the laws of physics and mechanics.

Consider the screen of your computer as a 2–dimensional plane. To make an object move in 2 dimensions, you simply need supply a series of X and Y coordinates. Coordinate X usually represents horizontal position and Y is usually vertical position. Constantly changing the combination of these two positions will result in the illusion of motion. Calculating X and Y is a task determined by what pattern of motion you desire.

Calculating the path of a projectile can be done in one of two ways: the hard way and the easy way. The hard way would be probably end up as a collage of imaginative calculations that somehow produce a fairly accurate simulation. The easy way is the logical way. In any book of physics or mechanics you'll find just about every formula for plotting the path of an object that is directly affected by a forward velocity, an upward velocity, and gravity – a projectile.

Forward Velocity

Every moving object on Earth has a forward velocity. Even if it only goes straight up, then straight down, it has a forward velocity. Of course this would be a forward velocity of zero.

Distance = Velocity * Time

Velocity is represented as some unit of distance, per some unit of time (eg. 10 feet/second). Multiplying by time cancels the time units. On the computer, the units of distance will be a column on the screen or an X-coordinate for a sprite.

The units of time could be obtained from the internal clock, but this imposes certain unnecessary complications. For one, the lowest unit is seconds which is an awfully long time unless the velocity too is very low. (We could use the TOD clock in the CIA but that would limit this demonstration to CIA equipped machines and using TI isn't all too portable either) Another, when the seconds reach 59, it is up to the programmer to add the minutes times 60 which steals processor time we may need.

There are probably more but the solution is simple: simulate time with a simple FOR/NEXT loop. This offers several synthetic advantages. You can express time in any unit such as tenths of seconds or even 3rds of seconds if you wish. Also, this avoids the potential for losing time since the clock will not increment until you have used the current "value" of time for your calculation, and subsequently used the results of that calculation for the plot. Further, simulated time can be generated within any chosen limits to suit the size and scale of the plotting surface. And unlike the clock, simulated time can be frozen.

So far our formula will look like this:

```
100 fv = 10 : rem forward velocity
120 for t = 0 to 159 step .2
130 x = fv * t
160 gosub 8000 : rem plot a point
180 next
```

The subroutine at 8000 is a plotting routine by Paul Higginbottom from a previous Transactor article. Note: please see Pro-

gram 1 at the end of the article – the programs presented in this text are meant to show the mechanics of our objective. Program 2 and 3 are the same simulation using sprites. The line numbers are different and the task of plotting is much simpler, but the mechanics are identical.

As you can see, time will be incremented from 0 to 159 in steps of 0.2, simulating a fifth of a second clock. "0 to 159" reflects the number of "half columns" available to the quarter–square plotting routine on an 80 column screen. The X coordinate is calculated and delivered to a subroutine that plots the coordinate on the screen, or the X-coordinate of sprite 0.

But no upward velocity has been given. In this case the projectile will simply move horizontally until the clock stops.

Upward Velocity and Gravity

This is the next element of the path of a projectile. It too is represented in distance per time unit, but unlike forward velocity, it is affected by the phenomena of Gravity.

Gravity is a unit of acceleration. When you drop an object, it starts with a velocity of zero and accelerates. Gravity is usually given as –32.2 feet per second squared. Different locations on Earth have gravitational constants slightly different than this depending on height above/below sea level, etc., but we'll use the natural constant for now. Further, if you go up high enough to drop your object, it will accelerate to a maximum velocity of about 119 mph, but we won't be doing that either.

The formula for our Y coordinate becomes the upward velocity multiplied by time, minus the effect of gravitational pull:

$$Y = UV*T - {}^1/_2G * T^2$$

The program becomes:

```
100 fv = 10 : rem forward velocity
110 uv = 45 : rem upward velocity
120 for t = 0 to 159 step .2
130 x = fv * t
140 y = 5 + uv * t – .5 * 32.2 * t↑2
160 gosub 8000 : rem plot a point
180 next
```

The value 5 at the beginning of line 140 is an initial height which gives the projectile a "floor" to bounce on. This brings us to the next consideration.

Impact and Decaying Velocity

As you well know, what goes up must come down. When our object hits the Earth, it will bounce, unless it's made of wet cement. Upon impact the object loses some of its initial upward velocity. Technically this is referred to as decay or a damping factor. However, our program poses another problem in its present form.

When the object impacts, there is a brief moment when it comes to a complete stop. Logically, time has become zero again and the formula repeats itself at the new decayed upward velocity. But our program shows time always incrementing. Therefore we must have a method of resetting the clock when the new cycle begins. The FOR/NEXT loop is the target for our next modification. It will now be used to represent time in the horizontal or X direction only. Vertical time will be stored in the variable T so we can reset Y time without resetting X time. This is not cheating – it merely makes the task simpler.

We need also know when the object impacts. Inotherwords, where is ground. Since we started from an initial height of 5, we'll say that ground is at 5. So when our calculation for Y yields a result less than 5, we know the object has bounced. This is also the point at which the decay takes effect. In this example, only upward velocity will decay – the forward velocity of a bouncing object is shown to be fairly constant, although you could impose damping on it too if you wish. The program becomes:

```
100 fv = 2  : rem forward velocity
110 uv = 45 : rem upward velocity
120 for j = 0 to 159
130 x = fv * j
140 y = 5 + uv * t – .5 * 32.2 * t↑2
150 if y<5 then y = 5 : t = 0 : uv = uv*.9
160 gosub 8000 : rem plot a point
170 t = t + .2
180 next
```

Notice that FV has been changed to 2 in line 100. This simply allows more cycles on the screen to show the effect of impact. Line 150 says if Y is less than 5 then Y equals 5. This little bit of cheating makes the ball bounce at the same vertical spot on the screen each time. Time is reset to zero and upward velocity is reduced by 10 percent.

Summary

With the program now in its final form, several possibilities exist. You can vary gravity slightly to show the effects of impact at different spots on Earth, or vary it a lot to simulate gravity on other planets. Your starting point does not necessarily have to be the same as the point of impact when the object comes down – you might project your object from some much higher elevation (eg. a cliff). Also, the object might not be the bouncing type – exploding objects don't usually bounce. If your object is the type to bounce, try different decay values for objects made from different materials. Depending on how hard they bounce

might affect the forward velocity too – something to think about.

Remember one thing most – computer simulations are all too often a task of logical thinking. The actions and reactions of a real physical object are usually the best way to simulate that object. Think first, program later.

Program 1 Portability Notes

The program has been set up for 80 column machines. Line 9010 prints two HOMEs to clear any windows, a clear screen, followed by a set graphics mode – CHR$(142). For 40 column PET/CBMs simply change LN=80 to LN=40 in line 9050.
As it stands, the program will leave a trail along the path of the projectile. To remove the trail, add:

```
8005 poke bs+p, 32
```

Line 8005 will poke a space into the previous POKE position thus erasing whatever was there. This is all that is required for all machine models.

VIC 20, Commodore 64, and C128 users (in 64 mode) will need to make changes in the setup subroutine at 9000.

C64
Change : 9010 print "[S]"
: 9050 ln=40 : bs=1024+24*ln : . . .
Add : 9055 cs=55296+24*ln
: 8025 poke cs+p, 1

Line 8025 is necessary for Kernal 2 C64s. POKing to the screen must be followed by a simultaneous POKE to colour memory or the characters will not show up.

VIC20
Change : 9010 print "[S]"
: 9050 ln=22 : bs=7680+22*ln : . . .
Add : 9055 cs=38400+22*ln
: 8025 poke cs+p, 2

VIC20 with expansion
Change : 9010 print "[S]"
: 9050 ln=22 : bs=4096+22*ln : . . .
Add : 9055 cs=37888+22*ln
: 8025 poke cs+p, 2

Since VIC 20 screens have only 23 lines, it will also be necessary to adjust the number 50 at the end of lines 410 and 430 to 46 (number of lines times 2). You will also nedd to change the first two numbers in the calculation for Y2 at lines 230 and 330. Start with:

```
Y2 = 23 + 22 * . . .
```

Warning: The plotting routine does not check to see if the POKE value is outside of screen memory. The potential for POKing into BASIC text space exists! Make sure you SAVE your program before trying new functions.

All the plotting efforts are performed by the two subroutines at 3000 and 8000. Subroutine 3000 plots a line from X1,Y1 to X2,Y2 by using subroutine 8000 to plot the points between the two coordinate pairs. When sub 8000 goes to plot a point, it must first determine if the target character space already contains a graphic. If it does, the new point must not interfere with the existing point in that space.

Question: How would you make the ball bounce off the right hand edge of the screen (ie. a wall). Hint: if x>79 then x=160–x. Use FV=7:UV=55.

C128 Notes

C128 users (in 128 mode) will have trouble adapting just about any program that POKEs to the screen. If preliminary information is correct, it seems the screen from the 128 is not memory addressable with just a POKE. More in a future Transactor.

Program 1

```
50 gosub 9000
500 rem ****    bouncing ball    ****
510 fv=2 : uv=55 : y1=1 : g=-32.2 : dc=.9
515 rem try fv=15 : uv=45. also fv=0.2
520 for j=0 to (ln*2-1)/fv
530 x=fv*j
540 y = y1 + uv * t + .5*g * (t↑2)
550 if y<y1 then y=y1 : t=0 : uv=uv*dc
560 gosub 8000 : rem plot a point
570 t=t+.2
580 next
590 end
8000 rem **********  plot x, y  **********
8010 tx=int(x+ir):ty=int(y+ir) :sq=am(tx and am, ty and am)
8020 p=tx/dv-int(ty/dv)*ln : poke bs+p, c(i(peek(p+bs))orsq)
8030 return
9000 rem **********    setup    **********
9010 print "[ssS]" chr$(142);
9020 dim c(15), i(255), am(1,1)
9030 for i=0 to 15 : read c(i) : i(c(i))=i : next
9040 for i=0to1 : for j=0to1 : am(j,i)=(j+1)*4↑i : nextj,i
9050 ln=80 : bs=32768+24*ln : dv=2 : am=1 : ir=.5
9060 data  32, 123, 108,  98, 126,  97, 127, 252
9070 data 124, 255, 225, 254, 226, 236, 251, 160
9080 return
```

Program 2 and 3 Notes

Chris has eliminated the FOR/NEXT loop from the procedure. Although distance is calculated differently, the result is the same. The FOR/NEXT loop incorporates screen boundary information. Here, XMAX is tested for the screen limits - once again, same result.

Program 2 comes in two parts. Part 1 (lines 10 to 30 and the data) merely creates a ball sprite in the cassette buffer. Enthusiasts might consider incorporating several ball sizes to add a depth dimension. A perspective factor would affect not only the size of the ball, but the ground level, the maximum X-coordinate, and the forward velocity as seen by the observer.

Program 3 is identical, except it creates the sprite using a little Zamara synthesis.

Program 2

```
ML  10 rem* create ball sprite at 896 *
MK  20 for i=896to959:read a:poke i,a:next
OB  30 end
HM  1000 data   0,   0,   0,   0,   0,   0,   3, 254
ON  1010 data   0,  15, 255, 128,  63, 255, 224,  63
JC  1020 data 255, 224, 127, 255, 240, 127, 255, 240
DD  1030 data 127, 255, 240, 127, 255, 240, 127, 255
AB  1040 data 240, 127, 255, 240, 127, 255, 240,  63
NG  1050 data 255, 224,  63, 255, 224,  15, 255, 128
DA  1060 data   3, 254,   0,   0,   0,   0,   0,   0
MP  1070 data   0,   0,   0,   0,   0,   0,   0,   0
```

```
GH  100 rem* bouncing ball simulation *
CO  110 :
FI  120 ymax=233: xmax=344
LA  130 fv=1          :rem forward velocity
MN  140 uv=100        :rem initial upward velocity
BO  150 y1=0          :rem y start position
NG  160 x=25          :rem x start position
OI  170 g=-32.2       :rem gravity in feet/s/s
KE  180 dc=.9         :rem elasticity of ''ball''
KH  190 t=0           :rem time starts at 0
MD  200 :
DE  210 vic=53248     :rem vic video chip
IH  220 poke vic+21,1 :rem enable sprite 0
HK  230 poke 2040,14  :rem sprite shape
MN  240 sx=vic: sy=vic+1: xhi=vic+16
OG  250 :
NN  260 rem-- main loop --
MG  270 x=x+fv
JI  280 y=y1 + uv*t +.5*g*(t*t)
DC  290 if y<y1 then y=y1:t=0:uv=uv*dc
JC  300 poke sx,x and 255:poke xhi,-(x>256)
DE  310 poke sy,ymax-y
KO  320 t=t+.2
JM  330 if x<=xmax then 270
EF  340 end
CN  350 :
```

Program 3

```
GH  100 rem* bouncing ball simulation *
CO  110 :
NI  120 ymax=237: xmax=327
NE  130 fv=1.4        :rem forward velocity
MN  140 uv=100        :rem initial upward velocity
BO  150 y1=0          :rem y start position
EG  160 x=10          :rem x start position
OI  170 g=-32.2       :rem gravity in feet/s/s
KE  180 dc=.9         :rem elasticity of ''ball''
KH  190 t=0           :rem time starts at 0
MD  200 :
GG  210 gosub 410 'create sprite shape
AF  220 :
HF  230 vic=53248     :rem vic video chip
MI  240 poke vic+21,1 :rem enable sprite 0
LL  250 poke 2040,14  :rem sprite shape
AP  260 sx=vic: sy=vic+1: xhi=vic+16
CI  270 :
MI  280 :
LP  290 rem-- main loop --
KI  300 x=x+fv
HK  310 y=y1 + uv*t +.5*g*(t*t)
BE  320 if y<y1 then y=y1:t=0:uv=uv*dc
HE  330 poke sx,x and 255:poke xhi,-(x>256)
BG  340 poke sy,ymax-y
IA  350 t=t+.2
OM  360 if x<=xmax then 300
CH  370 end
AP  380 :
KP  390 :
NO  400 rem** create sprite shape at 896 **
MP  410 fori=896to959:pokei,0:next
FG  420 fori=925to935step3:reada:pokei,a:next
OD  430 data 24,126,126,24
EN  440 return
```

SID's Programmable Filter

Karel Vander Lugt
Sioux Falls, SD

the filter in one SID chip may not be identical to that in another

Introduction

In recent years many articles and several books (ref 1–3) have been written on programming Commodore's SID (Sound Interface Device) chip. These, along with Commodore's Reference Guide (ref 4), do a good job of explaining most of the features of this versatile chip. However, there is one aspect of the 6581 chip which has received very little attention – its programmable filter. This is surprising since in reference 4 we read that, "The filter is, perhaps, the most important element in SID as it allows the generation of complex tone colors via subtractive synthesis."

Since the information needed to intelligently program SID's filter could not be found in the existing literature, the author conducted a series of tests. The object of these tests was to determine the exact effect of sending particular numbers to the four registers controlling SID's filter. This article summarizes the results and is intended to enable programmers to better utilize the full potential of the SID chip. Unfortunately, it was also found that the filter in one SID chip is not identical to that in another. This is discussed further in the article.

Testing The Filter

The equipment used to study SID's filter included an audio oscillator, a digital voltmeter, a frequency counter, a dual trace oscilloscope and several Commodore 64 computers. An external oscillator was used rather than one of the three internal voices because the internal voices only go up to 4 kHz and do not produce a pure sine wave.

The output of the oscillator, which is maintained at a constant amplitude, is connected to the Audio Input of the 5–Pin Audio/Video cable. The voltmeter reads the RMS value of the Audio Output, which is the signal going to the monitor. The frequency counter provides accurate frequency measurements and the oscilloscope permits simultaneous viewing of the input and output signals.

The first step in the procedure is to POKE a selected set of numbers into the four filter control registers on the SID chip. An 11–bit number in register 21–22 selects a cutoff frequency. The upper half of register 23 selects a resonance. Bit 3 of register 23 is left on so that the external signal from the oscillator goes through the filter. The lower three bits of register 23 remain off since the internal voices are not used. The volume is set to maximum with the lower four bits of register 24. The filter mode (low pass, high pass, or band pass) is selected with bits 4 through 6 of register 24.

After the filter was programmed for a particular cutoff frequency, resonance and mode, the RMS output voltage as a function of the input frequency was obtained. About 30 readings, ranging from 10 Hz to 20 kHz, were taken for each set of data.

For each of the three filter modes, several sets of data were obtained using different numbers in SID's registers controlling the cutoff frequency and the resonance. Each set of data was plotted with the y–axis as the normalized output in decibels and the x–axis as the log of the frequency. The results are discussed in the next section.

Results

A. The Three Filter Modes

An ideal low pass filter passes all frequencies below the cutoff frequency and severely attenuates all those above it. A typical response for SID's low pass filter is shown in Figure 1. The y–axis is the output signal in decibels (y = 20*log(Vout/Vmax)). The x–axis is the frequency of the input signal, also on a logarithmic scale. The resonance was maximum (15) for the upper curve and minimum (0) for the lower curve. In both cases, the number 800 was placed in register 21–22.

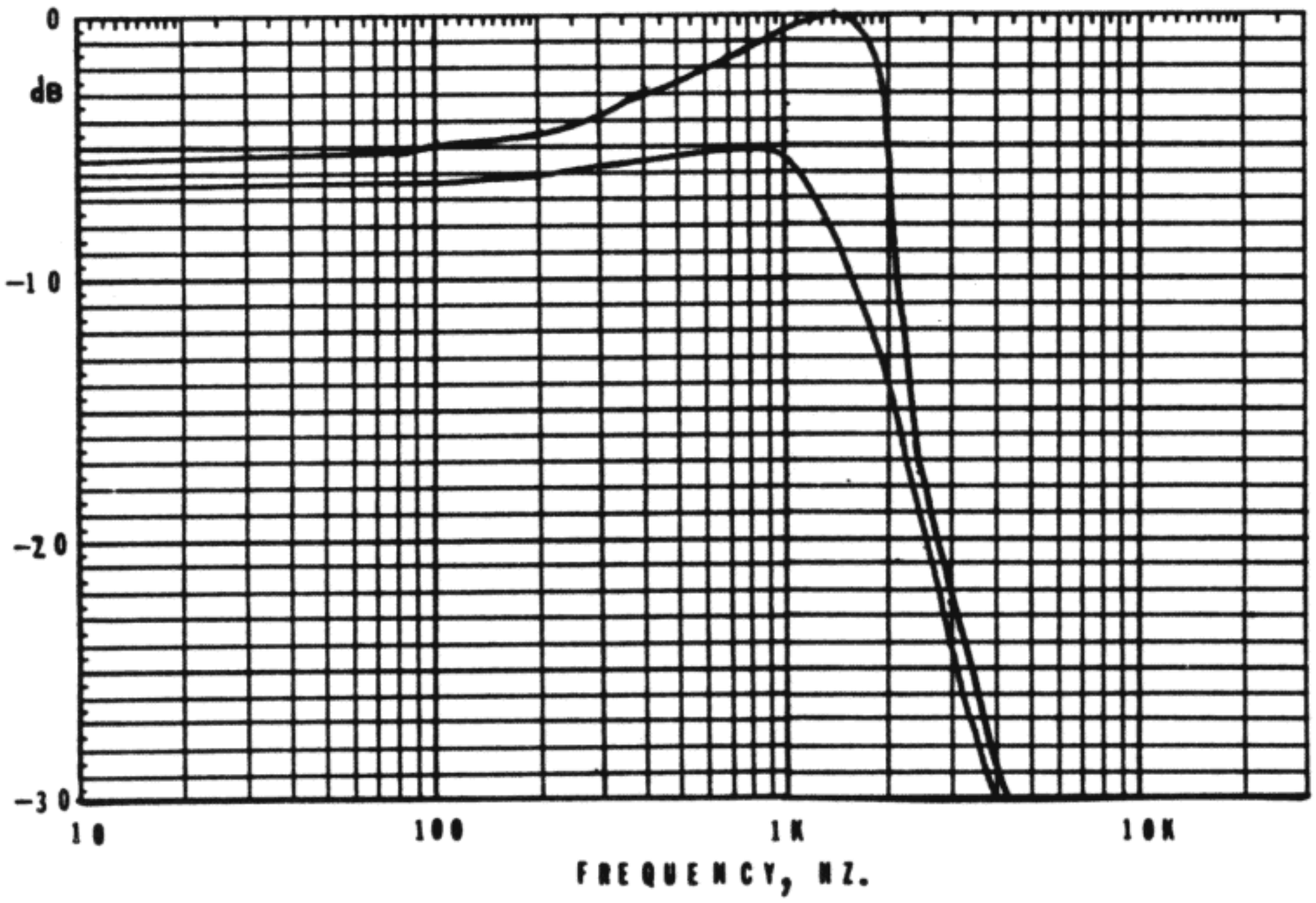


Figure 1: *The SID filter in its low pass mode. The upper curve is for maximum resonance(15) and the lower curve is for minimum resonance(0).*

The cutoff frequency is determined by the –3dB point, that is, the frequency at which the signal is attenuated to 70% of its maximum value. For this particular C–64, a setting of 800 in register 21–22 produces a low pass cutoff frequency of 1840 Hz when the resonance is 15 and a low pass cutoff frequency of 1360 Hz when the resonance is zero.

With the resonance set at zero, frequencies above the cutoff point are attenuated at a rate of approximately 14dB per octave. As seen from the upper curve, the rolloff rate is even higher for a resonance of 15.

Figure 2 illustrates a typical response for SID's high pass mode. The upper curve is for a resonance of 15 and the lower curve is for a resonance of zero. As before, the number placed in register 21–22 was 800. The –3dB cutoff frequency is 1115 Hz for the upper curve and 865 Hz for the lower curve. The rate of attenuation below the cutoff point is approximately 8dB per octave.

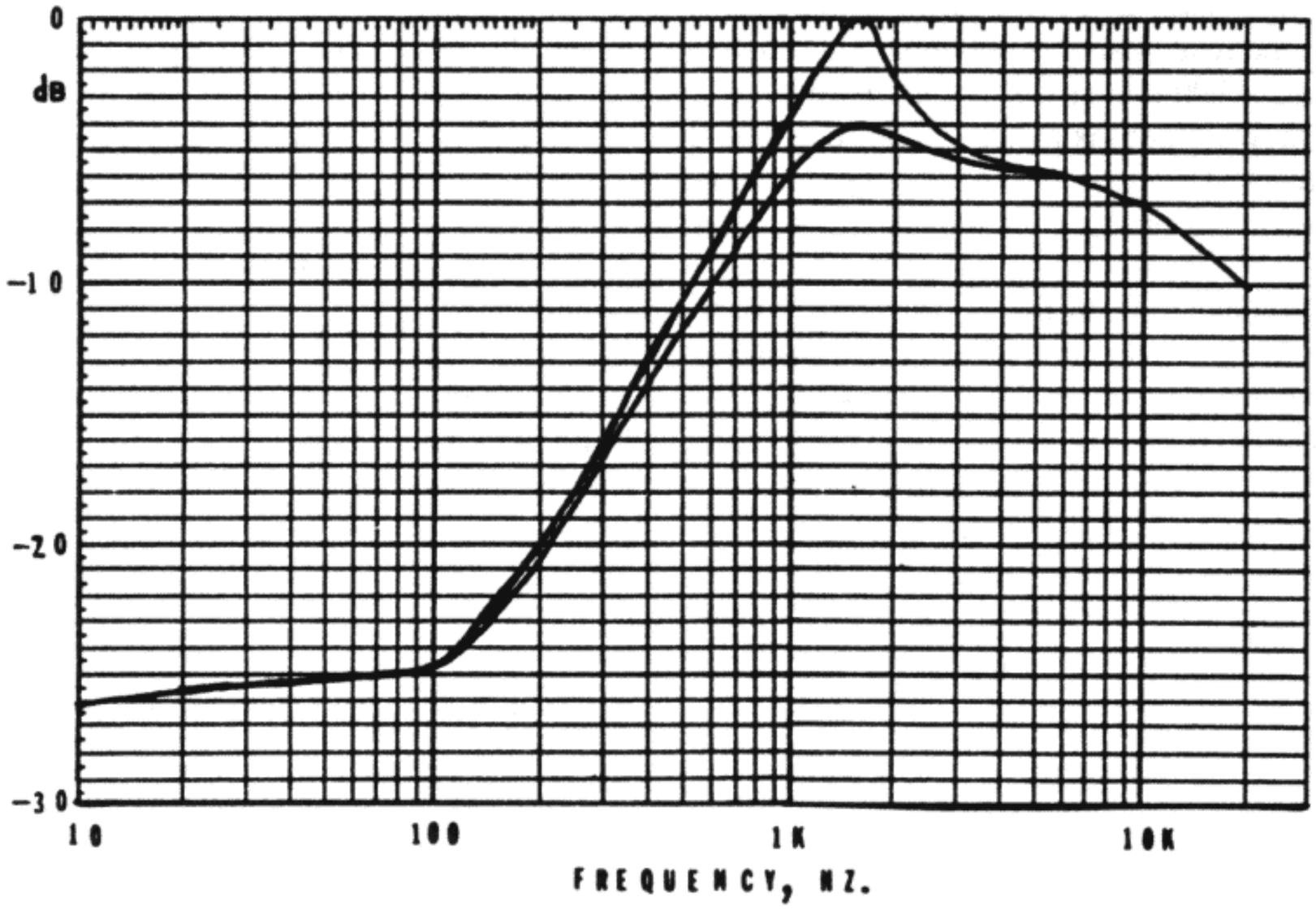


Figure 2: *The SID filter in its high pass mode. The upper curve is for maximum resonance and the lower curve is for minimum resonance.*

A typical response for the band pass mode is shown in Figure 3. The upper curve is for a resonance of 15. The low frequency cutoff point is 890 Hz and the high frequency point is 1900 Hz. Thus the bandwidth is 1010 Hz. The center frequency (the square root of the product of the two cutoff frequencies) is 1300 Hz. The Q value (center frequency/bandwidth) is 1.29.

When the resonance is set to zero, the peak is broader and the cutoff points shift to lower frequencies. This is shown by the lower curve in Figure 3. With the resonance at zero, the low frequency cutoff is 520 Hz and the high frequency cutoff is 1910 Hz. The bandwidth is 1390 Hz. The center frequency is 997 Hz and the Q value is reduced to 0.72.

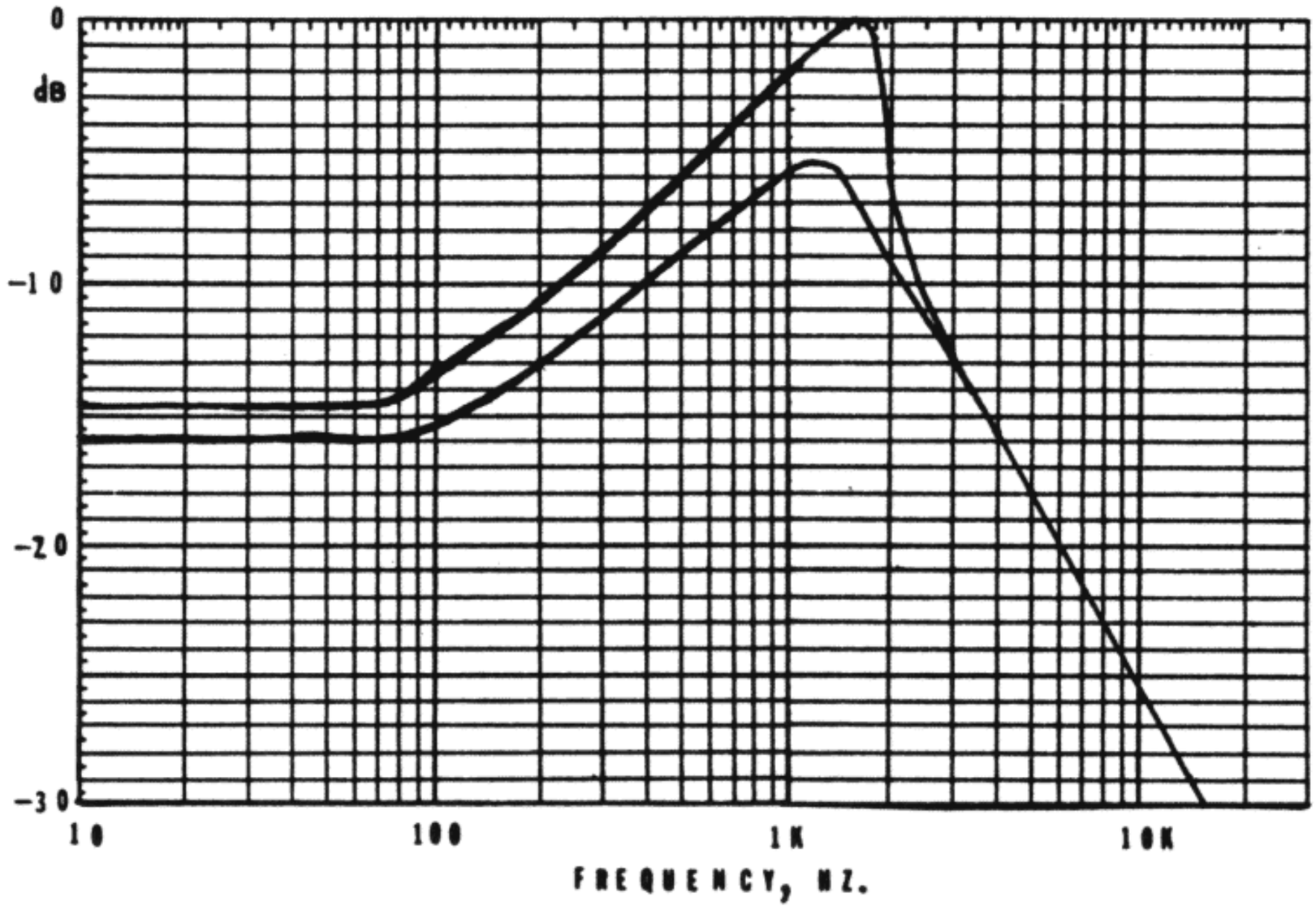


Figure 3: *The SID filter in the band pass mode. Again, the upper curve is for a resonance of 15 while the lower curve is for a resonance of 0.*

In both cases the rate of attenuation is approximately 4dB per octave on the low frequency side and about 7dB per octave at high frequencies. With the resonance at 15, the rolloff rate near the upper cutoff point is considerably higher, about 25dB per octave.

B. The Cutoff Frequency

If one wants to use SID's filter to modify sounds, it is important to know how the cutoff frequency can be adjusted with software. The literature suggests the cutoff frequency can be adjusted between approximately 30 Hz and 12 kHz by proper choice of the number sent to register 21–22. Some sources (ref 5) state the relationship is linear while others (ref 2) imply it is exponential. The results of this study indicate the relationship is neither linear nor exponential and that the range varies greatly from one SID chip to another.

Figure 4 shows, for one particular Commodore 64, how the –3dB point varies with the setting of register 21–22. The resonance is set at zero. The upper curve is for the low pass mode and the lower curve is for the high pass mode. Over the range of register 21–22 (0

to 2047) the low pass cutoff point varies from 78 Hz to 7595 Hz. The high pass cutoff point varies from 55 Hz to 4935 Hz. In either case, the relationship is not linear over the entire range. Below a setting of 800 the curves are approximately parabolic and above 800 they are approximately linear, but with different slopes and intercepts. Using a curve fitting program it was found that the data can be fit reasonably well with cubic polynomials.

Unfortunately, when the same series of tests was done using a different C–64, a different result was obtained. In fact, there was considerable variation among the several C–64s used. Figure 5 illustrates how the high pass cutoff point varies with the setting of register 21–22 for three different Commodore 64s. The variation is disappointingly large. For example, in one computer (lower curve) a setting of 1500 in register 21–22 produces a high pass cutoff frequency of 3365 Hz while in another computer (upper curve) the same setting produces a high pass cutoff frequency of 11,000 Hz. The low pass and band pass cutoff frequencies also exhibit these large variations.

Conclusion

The SID chip is often touted for its excellent sound producing capabilities. Much of the praise is deserved. Commodore's incorporation of a programmable filter into the chip is laudable; however, until more consistency can be obtained in the setting of the cutoff frequencies via register 21–22, the filter is of limited use to programmers.

Dr. Vander Lugt can be contacted at the following address:

Dr. K. Vander Lugt
Professor of Physics
Augustana College
Sioux Falls, SD 57197
(phone 605 336–4911)

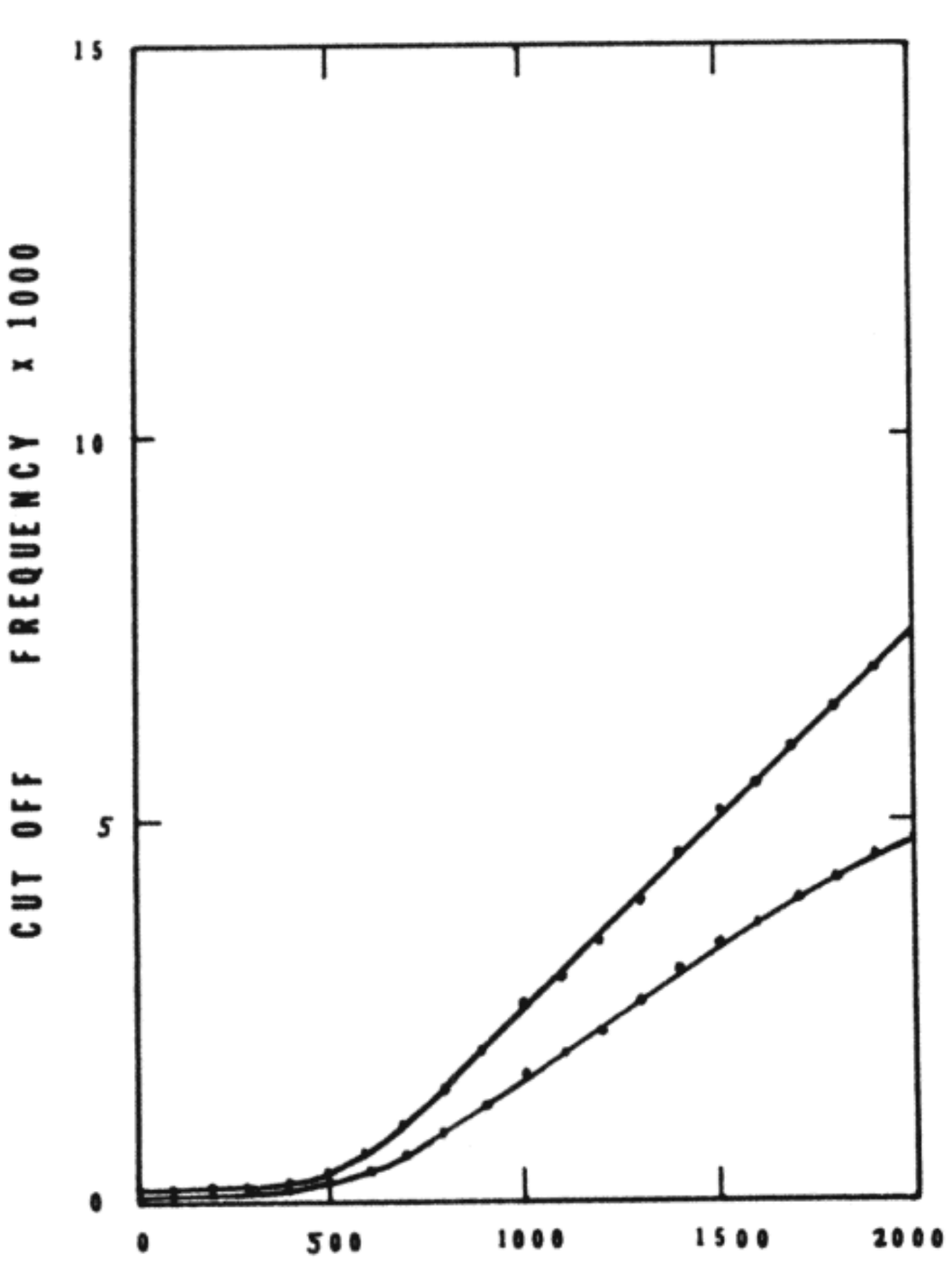


Figure 4: *Relationship between cutoff frequency and the number POKEd to SID's cutoff frequency register. The upper curve is for the low pass mode and the lower curve is for the high pass mode. The cutoff points for the band pass mode fall between these curves. The data in Figures 1,2 and 3 was taken with the number 800 POKEd into SID register 21–22.*

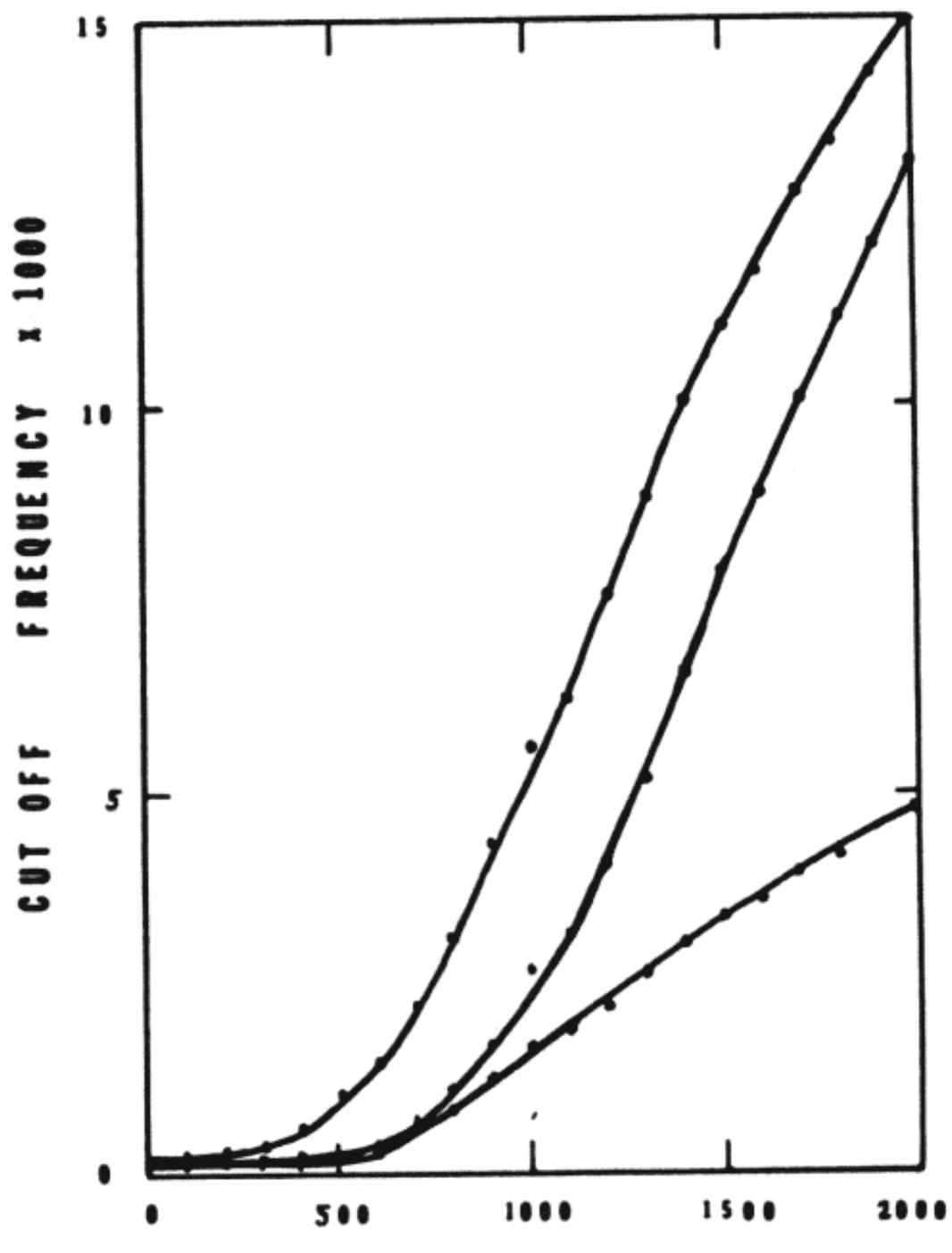


Figure 5: *Relationship between high pass cutoff frequency and register setting for three different Commodore 64s. Unfortunately, the variation among SID filters is large.*

REFERENCES

1. Vogel, James and Nevin B. Scrimshaw, "The Commodore 64 Music Book" (Boston,MA:Birkhauser).
2. Heilborn, John, "COMPUTE!'s Beginner's Guide to Commodore 64 Sound" (Greensboro,NC:COMPUTE! Publications,1984)pp154.
3. Behrendt, Bill L., "Music and Sound for the Commodore 64" (Englewood Cliffs,NJ:Prentice–Hall,1983).
4. "Commodore 64 Programmer's Reference Guide" (Commodore Business Machines, 1982).
5. Leemon, Sheldon, "Mapping the Commodore 64" (Greensboro,NC:COMPUTE! Publications,1984)pp166.

The Compressor: A High–Resolution Picture Compressor/Decompressor

Chris Zamara, Technical Editor

Store your high–res pictures in less than half the usual disk space!

High–res pictures are nice to look at, but they eat up lots of disk space – to the tune of 8,000 bytes per picture, or 32 disk blocks. That can be a problem if you're trying to write a game with many different screens or an adventure with dozens of scenes – you'll quickly reach the 664 block limit of a 1541 disk! Not only that, but loading in a picture from the lumbering 1541 can take way too long – about 20 seconds. As you may have guessed by now, you're about to find a way out of both these problems: THE COMPRESSOR.

The Compressor will compress your pictures (either hi–res or multi–colour) by taking advantage of byte repetitions. This occurs often in pictures, for example when a large area is one solid colour, creating scores of consecutive zeroes or 255s in the bitmap. Any picture with vast blank or single–colour areas will benefit greatly by being compressed. An uncompressed picture takes 32 blocks on disk, but a typical compressed bitmap takes less than 20. The actual size of a compressed picture varies greatly depending on the pattern, from about 90 bytes for a blank screen up to over 20 for a full–screen, highly detailed display. Generally, the worst case for a picture – something like a portrait which fills the entire screen – will be compressed to about 20 blocks; that translates to about a 38 percent savings in disk space. (Actually, the extreme worst case is a screen filled with random bytes; it takes up about the same space compressed or uncompressed.) Better, though, are pictures with a pattern or outline drawing against a single–colour background. For example, there is a picture of Snoopy on one of the Toronto Pet User's group hi–res picture disks; Snoopy gets compressed to a mere 9 blocks! That means about a 72 percent savings in space, and also in time when the picture is brought in by the decompressor (which is part of the same program). The average size for compressed pictures seems to be about 15 blocks, less than half the usual space.

The Compressor was originally written to save a bitmap screen from memory to a disk file. That version can find use in applications where you've created a picture from BASIC or a hi–res graphics utility and wish to save it in compressed form. The save–from–memory version of The Compressor appears in BASIC form in Listing 2. In many applications, however, you already have a hi–res picture on a disk file, either from another source or created with a commercial drawing package like Koalapainter. To convert an existing hi–res picture, use the version of The Compressor in Listing 1. The file to be converted must be in LOADable form, i.e. it must be a PRG or SEQ file and contain the load address as the first two bytes. The compressed file created by The Compressor will be the same format, the load address telling the uncompressor where to put the picture when it is brought in. Picture files created with Koalapainter (Koala pad software) or Commodore's Animation Station can be directly compressed by The Compressor, but you'll probably want to separate the bit–map and colour–map information into separate files first. More on that later.

To use The Compressor, first create the machine–language program file on disk by entering the BASIC loader in Listing 1 or 2 (depending on your preference) and running it. This will put the file called "comp1.obj" or "comp2.obj" on disk which you can then bring into memory with a normal LOAD (using " ,8,1 " after the filename). The example program in Listing 3 loads "comp1.obj", so you'll have to create it before running listing 3. If you have the Transactor disk, the object file is already on disk and you can skip the above step.

Version 1: Compressing A Hi–Res Picture Program File

Compressing and de–compressing a picture file can be easily done using the example program in Listing 3. You can use this program to compress a picture or load a compressed picture, or you can use The Compressor from direct mode or in your own programs. To compress a picture file with The Compressor (the file–based one in Listing 1), you have to open the input and output files in BASIC first. The input file – the one on disk to be compressed – must be OPENed as file #8, and the new compressed file which is to be created must be file #9. For example, if you have a high–res picture stored on disk as a

program file called "design1" and you wish to make a compressed version called "design1/c", you could just use the following BASIC:

```
open 8,8,2, " design1 "
open 9,8,1, " 0:design1/c " : rem create PRG file
```

Then, to compress the file, just execute The Compressor with

```
sys 49152
```

Note: Notice the absence of 'comma P', 'comma R' and 'comma W' after the filenames in the OPEN statements? When only a filename is specified (SA 2 to 14), the DOS defaults to a Read of ANY file type, unless Secondary Address 0 is used – then it defaults to a Read of a PRG file. With Secondary Address 1 the default is ',p,w' thus eliminating the need for a filename suffix.

After a little while (1 to 2 minutes with a 1541 drive), the disk files will close and the computer will come back with the usual READY. You now have the compressed file "design1/c" on disk, which should be considerably smaller than the original "design1". It will give you the identical screen image, however, once you load it in with the de-compressor part of the program. To de-compress a picture, just OPEN the file for input as file #8 and SYS 49155, like this:

```
open 8,8,2, " design1/c "
sys 49155
```

The picture will then be put into memory at the start address specified by the first two bytes in the original file (low, high). In other words, the picture will go in the same place as the original uncompressed version would if LOADed directly from BASIC.

Version 2: Compressing A Picture From Memory

Use Listing 2 to create the memory-based version of The Compressor (or load " comp2.obj " ,8,1 from the Transactor disk). To use this version, you don't need to open any files from BASIC. Just supply the name of the file you wish to create, like this:

```
sys 49152, " filename "
```

The 8000 bytes of memory starting at $2000 (8192) will be saved in compressed form under the given filename. If you wish to save a picture from a different location, POKE the desired address in locations 49158 and 49159 in low, high format before doing the SYS. Since a high-res screen always falls on an 8k boundary, you will usually just POKE 49159 (address high byte) with a multiple of 32. For example, if your picture is in the RAM at $E000 (The Compressor always reads the RAM, not ROM), you would save it with:

```
poke49159,224          : rem high byte, location $e000
sys 49152, " filename "  : rem save picture as " filename "
```

The Listing 2 version also uses the above method for loading a compressed picture. Just use the SYS address 49155 instead of 49152, like this:

```
sys49155, " filename "     : rem load compressed picture
```

The above command will bring the picture "filename" into memory at the address it was originally at when saved (i.e. compressed).

"Koalapainter" and "Animation Station" Picture Files

To produce artistic-looking work on the 64, your best bet is to use a graphics tablet of some kind and a good graphics package. An inexpensive such system is the Koala Pad from Koala Technologies Corp., Which consists of the Koala pad touch tablet and Koalapainter software. Another excellent software package is Commodore's Animation Station, which can be used with the Koala Pad touch tablet. Pictures saved with either of these packages go on disk as a PRG file containing the bit-map and both colour maps for the multi-colour image. If you wish to use these pictures in your own program or just display them without using the graphics software, you can use the programs in listing 4 or 5 to split the picture file into three LOADable modules, which may then be compressed with The Compressor.

The BASIC programs "Koala split" and "Anim split" in Listings 4 and 5 allow you to select the start address of the bit-map and colour-map files which will be created from the original picture file saved by the graphics software. You can then LOAD these files directly from BASIC into whatever memory areas you specified and display the picture by setting up the right VIC-II video chip parameters. The idea, of course, is to use The Compressor on the 8000-byte bit-map picture file to free up valuable disk space and speed up retrieval of the picture.

The "Koala split" or "Anim split" program will ask for the start address of the hi-res picture and the colour map, which you must respond to in hexadecimal. Default values are provided which locate the picture at $E000 and the colour map at $CC00 (these are good, out-of-the-way spots). The other colour map resides in the 1k colour nybbles at $D800 and is not relocatable. The program then asks for the filename of the picture. The name used here is the one that is used *in the graphics software*, not the actual name on disk. For example, an Animation station picture called "design" would appear as "pi.design" in the disk directory; just use "design" as the filename. Similarly, Koala pad files are preceded by a CHR$(129), which appears as a reverse-A in the disk directory; just use the Koala name itself, eg. "pic a design", "pic b house", etc.

The split program takes quite a while to do its thing, since it has to copy over 10,000 bytes from one file to another, so better schedule a run to coincide with your next coffee break. Also,

make sure you have enough room on your disk to accommodate the three new files that will be created, requiring a total of 40 additional blocks. These files will be named with the filename you supplied as input followed by the extension ".pic", ".c1", or ".c2". The ".pic" file is the actual hi-res bitmap which you'll probably want to cut down to size by unleashing The Compressor on it. The ".c1" file is the relocatable colour map which resides at the chosen location. The ".c2" file has the start address of $D800, where it must load into to supply the third source of colour information for the picture (the first two sources come from the ".c1" file).

After running the split program, you have three files which you can LOAD into memory, and display if you wish by appropriate POKEs to VIC chip registers. To display a picture at $E000 with the colour map at $CC00, the BASIC would be:

```
poke53265,59:poke53272,63:poke53270,216:poke56576,0
```

To return to the normal text screen,

```
poke53265,27:poke53272,23:poke53270,200:poke56576,3
```

(See the article "VIC Parameters" in Volume 5, Issue 6 for more on setting up the VIC chip registers.) Also, don't forget to change the background colour at 53281 to the colour indicated by the last byte in the picture file.

How The Compressor Works

There's no profound or amazing tricks used here; it's a very straightforward approach that seems to work pretty well in practical use. The compressor just looks at each byte and compares it to the previous one. If it finds more than three bytes in succession which are the same, instead of storing that many bytes, it just stores a special control byte (arbitrarily 254) followed by the byte which is to be repeated and the number of repetitions. As an example, an uncompressed file containing these bytes:

```
10 196 202 15 15 15 15 15 15 15 15 15 15 15 32 76
```

Would be compressed as:

```
10 196 202 254 15 11 32 76
```

The first three bytes are copied verbatim, but the group of eleven 15s is represented by the control sequence '254 15 11'. Groups greater than 255 bytes long are represented by more than one control sequence. But what happens when a single 254 is encountered in the file? That is easily handled by the control sequence:

```
254 254 1
```

As with any other byte, up to 255 consecutive '254's are represented by a single control sequence. As you can see, a 254 all alone or in groups of two or three will use more memory when compressed than uncompressed. Fortunately, that doesn't happen often enough to be a concern.

The algorithm used to compress the data is quite simple: the bytes in the file to be compressed are read one by one. If the byte just read is the same as the one before it, a counter is incremented. If not, the current count is zeroed, then used to generate N repetitions of the previous byte, or if the count is greater than three, a control sequence is written (254, previous byte, number of repetitions). A control sequence must also be generated if the count ever exceeds 255. That's it! Not too tough a task, even in ML, but it gets the job done!

The decompressor is even simpler. It just reads a byte and stores it unchanged if it isn't a 254. If it is, the next two bytes are fetched and the given byte is copied into the next N memory locations. The process repeats until the end of file is reached.

Other Applications

So far, you've seen that the compressor can be used with ordinary high-res or multi-colour pictures, even those created by commercial graphics software. But really, there's nothing that says The Compressor can only compress picture information. Any data that are stored as a SEQ or PRG file on disk and are likely to contain repetitions of a single byte can benefit by being compressed. BASIC programs are not very good candidates for compression, but a long text file containing many spaces, or a sequential database with many blank records might be. Sprite definition files are perfect.

Using The compressor with high-res pictures, though, adds a new dimension to graphics. The smaller and less detailed your pictures are, the less space they'll take on disk. In other words, there's a linear relationship between the physical size of a picture on the screen, and the size of it in terms of disk blocks – providing that the unused portions of the screen are blank. With the compressor at your disposal, you may be more willing to use small pictures or simple sketches in situations where an entire 32 block picture file would be just too expensive (in terms of disk space and loading time) to be worthwhile. After all, just because a picture is worth a thousand words doesn't mean it has to take up 8,000 bytes.

Listing 1: BASIC program to create "comp1.obj" file on disk. This version will create a compressed disk file from an existing file. The first two bytes of the file must be the load address. To use the program, load "comp1.obj",8,1, then OPEN file #8 as the input file and file #9 as the output file. SYS 49152 to compress the file. To load a compressed file, open the file as #8 for input, then sys49155.

```
AK 10 rem* file creator for "comp1.obj" *
LI 20 cs=0
BG 30 for i=49152 to 49418:read a:cs=cs+a:next
IE 40 if cs<>37827 then print "!data error!" : stop
KA 50 open 1,8,2,"@0:comp1.obj,p,w"
FJ 60 print#1,chr$(0)chr$(192);: rem $c000
OF 70 restore
AI 80 for i=1 to 267:read a:print#1,chr$(a);:next
NC 90 close 1
IN 100 :
FB 1000 data  76,  10, 192,  76,  40, 192,   1,   0
LB 1010 data   0,   0,  32, 243, 192,  32, 228, 255
LH 1020 data  32, 255, 192,  32, 210, 255,  32, 243
KL 1030 data 192,  32, 228, 255,  32, 255, 192,  32
NE 1040 data 210, 255,  32,  72, 192,  76,  58, 192
KD 1050 data  32, 243, 192,  32, 228, 255, 133, 251
MH 1060 data  32, 228, 255, 133, 252, 160,   0,  32
JJ 1070 data 200, 192,  32, 204, 255, 169,   8,  32
LJ 1080 data 195, 255, 169,   9,  32, 195, 255,  96
HG 1090 data  32, 243, 192,  32, 228, 255, 141,   8
EJ 1100 data 192,  32,  93, 192, 173,   9, 192, 240
JA 1110 data 248,  32, 174, 192,  96,  32, 243, 192
GP 1120 data  32, 228, 255,  72,  32, 183, 255, 141
II 1130 data   9, 192, 104, 141,   7, 192, 205,   8
CJ 1140 data 192, 208,  14, 238,   6, 192, 208,   6
GJ 1150 data 206,   6, 192,  32, 174, 192,  76, 167
KJ 1160 data 192, 173,   6, 192, 201,   6, 176,  28
DP 1170 data 170, 173,   8, 192, 201, 254, 240,  20
LO 1180 data  32, 255, 192, 173,   8, 192,  32, 210
MK 1190 data 255, 202, 208, 250, 169,   1, 141,   6
FC 1200 data 192,  76, 167, 192,  32, 174, 192, 173
BA 1210 data   7, 192, 141,   8, 192,  96,  32, 255
PJ 1220 data 192, 169, 254,  32, 210, 255, 173,   8
KB 1230 data 192,  32, 210, 255, 173,   6, 192,  32
EO 1240 data 210, 255, 169,   1, 141,   6, 192,  96
LD 1250 data  32, 228, 255, 201, 254, 240,   6,  32
GG 1260 data 234, 192,  76, 228, 192,  32, 228, 255
EF 1270 data  72,  32, 228, 255, 170, 104,  32, 234
BB 1280 data 192, 202, 208, 250,  32, 183, 255, 240
JN 1290 data 223,  96, 145, 251, 230, 251, 208,   2
LC 1300 data 230, 252,  96,  72, 138,  72, 162,   8
LP 1310 data  32, 198, 255, 104, 170, 104,  96,  72
HF 1320 data 138,  72, 162,   9,  32, 201, 255, 104
BJ 1330 data 170, 104,  96
```

Listing 2: BASIC program to create machine-language program "comp2.obj" on disk. "comp2.obj" will compress a high-res picture in memory (default location $2000) and store it on disk under the specified filename.

```
DK 10 rem* file creator for "comp2.obj" *
MI 20 cs=0: q$=chr$(34)
CG 30 for i=49152 to 49493:read a:cs=cs+a:next
GC 40 if cs<>45400 then print "!data error!" : stop
NG 41 print "{q}to compress pic at $2000:
BM 42 print "   sys49152,"q$"filename
BG 43 print "{q}to load compressed picture:
OO 44 print "   sys49155,"q$"filename"
JL 50 open 1,8,1,"@0:comp2.obj"
FJ 60 print#1,chr$(0)chr$(192);: rem $c000
OF 70 restore
GH 80 for i=1 to 342:read a:print#1,chr$(a);:next
NC 90 close 1
IN 100 :
EI 1000 data  76,  15, 192,  76,  24, 192,   0,  32
LN 1010 data   1,   0,   0,   0,   0,   0,   0, 169
JF 1020 data   1, 141,  11, 192, 160,   1, 208,   7
KM 1030 data 169,   0, 141,  11, 192, 160,   2, 165
OB 1040 data   1, 141,  12, 192, 169,   8, 170,  32
KH 1050 data 186, 255,  32, 253, 174,  32, 158, 173
DF 1060 data  32, 143, 173, 160,   0, 177, 100,  72
MD 1070 data 200, 177, 100, 170, 200, 177, 100, 168
KH 1080 data 104,  32, 189, 255,  32, 192, 255, 162
GG 1090 data   8, 160,   0, 173,  11, 192, 240,  21
KH 1100 data  32, 201, 255, 173,   6, 192,  32, 210
MG 1110 data 255, 173,   7, 192,  32, 210, 255,  32
JN 1120 data 126, 192,  76, 117, 192,  32, 198, 255
BM 1130 data  32, 228, 255, 133, 251,  32, 228, 255
FO 1140 data 133, 252,  32,  42, 193,  32, 204, 255
HN 1150 data 169,   8,  32, 195, 255,  96, 173,   6
LB 1160 data 192, 133, 251,  24, 105,  64, 141,  13
IA 1170 data 192, 173,   7, 192, 133, 252, 105,  31
BK 1180 data 141,  14, 192,  32,  24, 193, 141,  10
NB 1190 data 192, 160,   1,  32, 179, 192, 165, 252
KE 1200 data 205,  14, 192, 208,   8, 165, 251, 205
NH 1210 data  13, 192, 144, 239,  96, 144, 236,  32
OH 1220 data   6, 193,  96,  32,  24, 193, 141,   9
NB 1230 data 192, 205,  10, 192, 208,  19, 238,   8
BB 1240 data 192, 208,  11, 206,   8, 192,  32,   6
DC 1250 data 193, 169,   1, 141,   8, 192,  76, 249
GP 1260 data 192, 173,   8, 192, 201,   4, 176,  25
DE 1270 data 170, 173,  10, 192, 201, 254, 240,  17
NP 1280 data 173,  10, 192,  32, 210, 255, 202, 208
OD 1290 data 247, 169,   1, 141,   8, 192,  76, 249
NL 1300 data 192,  32,   6, 193, 169,   1, 141,   8
EJ 1310 data 192, 173,   9, 192, 141,  10, 192, 230
FI 1320 data 251, 208,   2, 230, 252,  96, 169, 254
JK 1330 data  32, 210, 255, 173,  10, 192,  32, 210
BH 1340 data 255, 173,   8, 192,  32, 210, 255,  96
NB 1350 data 120, 165,   1,  41, 252, 133,   1, 177
DG 1360 data 251,  72, 173,  12, 192, 133,   1,  88
KF 1370 data 104,  96,  32, 228, 255, 201, 254, 240
NP 1380 data  11, 145, 251, 230, 251, 208,   2, 230
PK 1390 data 252,  76,  80, 193,  32, 228, 255,  72
DK 1400 data  32, 228, 255, 170, 104, 145, 251, 230
MF 1410 data 251, 208,   2, 230, 252, 202, 208, 245
DB 1420 data  32, 183, 255, 240, 213,  96
```

Listing 3: BASIC program which uses "comp1.obj" to compress or de-compress a hi-res picture file. The code illustrates how to use The Compressor

```
NP  100 rem* compress or de-compress    *
EG  110 rem* a high-res picture on disk  *
MO  120 :
MB  130 if a=0 then a=1:load "comp1.obj",8,1
PC  135 open 15,8,15: rem error channel
LL  140 print" Sqn Select one:
ED  150 print" q 1) Compress a picture file
ID  160 print" 2) Load in a compressed file
GE  170 get a$:if a$<>"1" and a$<>"2" then170
EP  180 rem bring in 'the compressor'
BF  190 input" q picture filename";f1$
LK  200 if a$="2" then 330
GE  210 :
HI  220 rem* create compressed file *
ED  230 input" name for new compressed file";f2$
AH  240 open 8,8,2,f1$
NI  250 gosub 400: rem check for disk error
EI  260 open 9,8,1,"0:"+f2$
BK  270 gosub 400: rem check for disk error
BG  280 sys 49152: rem compress file
LF  290 print" new picture file now on disk"
MC  300 end
KK  310 :
CE  320 rem* load compressed file *
KM  330 open 8,8,2,f1$
HO  340 gosub 400: rem check for disk error
PM  350 sys 49155: rem de-compress file
HJ  360 print" picture now in memory."
CH  370 end
AP  380 :
LP  390 rem* get disk status subroutine *
LL  400 input#15,a$,b$,c$,d$
PI  410 if val(a$)then print" qq disk error: "a$",
        "b$","c$","d$: end
AM  420 return
```

Listing 4: "Koala split". This program will take a Koalapainter picture file and create three LOAD-able program files which contain the bitmap and colour maps. The start address of the bitmap and one colour map is relocatable.

```
CM  100 rem* "koala split"
JM  110 rem* split a koalapainter picture
GA  120 rem* file into 3 loadable prg files
GP  130 :
HL  140 z$=chr$(0): open 15,8,15
JC  150 input" Sqqq start of hi-res picture[3 spaces]
        e000[7 lefts]";h$
CD  160 gosub 1000: if er then print" Q ";:goto150
IA  170 pl=l: ph=h
KO  180 input" start of colour map[7 spaces]
        cc00[7 lefts]";h$
AF  190 gosub 1000: if er then print" Q ";:goto150
FO  200 cl=l: ch=h
ID  210 input" q filename of koala file";f$
FJ  220 open1,8,12,left$(chr$(129)+f$+
        "[12 spaces]",15)
CP  230 gosub 2000: rem check for disk error
DL  240 get#1,a$,b$
PA  250 open2,8,11,"@0:"+f$+".pic,p,w"
AB  260 gosub 2000: rem check for disk error
FH  270 print#2,chr$(pl)chr$(ph);
GB  280 fori=1to8000:get#1,a$:print#2,left$(a$+z$,1);
        :next
IP  290 close2
KF  300 open2,8,11,"@0:"+f$+".c1,p,w"
CE  310 print#2,chr$(cl)chr$(ch);
ME  320 gosub 2000: rem check for disk error
DD  330 fori=1to1000:get#1,a$:print#2,left$(a$+z$,1);
        :next
KC  340 close2
OI  350 open2,8,11,"@0:"+f$+".c2,p,w"
CN  360 print#2,chr$(0)chr$(216);: rem colour nybbles
OH  370 gosub 2000: rem check for disk error
FG  380 fori=1to1000:get#1,a$:print#2,left$(a$+z$,1);
        :next
MF  390 close2
CG  400 print" qq The background colour is:";
LO  410 get#1,a$: printasc(a$+z$)
AA  420 close1: close15
OK  430 end
MC  440 :
FP  1000 rem* convert hex f$ to dec h,l *
FG  1005 er=0
JK  1010 if len(h$)<>4then er=1:return
IK  1020 d=0:fori=1to4:h=asc(mid$(h$,i))-48
        :d=d*16+h+7*(h>9):next
NO  1030 h=int(d)/256: l=d-h*256
MC  1040 return
OI  1050 :
FE  2000 rem* get disk status subroutine *
FA  2010 input#15,a$,b$,c$,d$
JN  2020 if val(a$)then print" qq disk error: "a$",
        "b$","c$","d$: end
KA  2030 return
```

Listing 5: "Anim split". Performs the same function as Koala split above, but for "Animation Station" files.

```
FI  100 rem*  " anim split "          *
BE  110 rem* converts a picture file  *
NL  120 rem* created by the animation *
LA  130 rem* station into 3 loadable  *
PO  140 rem* prg files.               *
KA  150 :
LM  160 z$ = chr$(0): open 15,8,15
ND  170 input " Sqqq start of hi-res picture[3 spaces]
        e000[7 lefts] " ;h$
OE  180 gosub 1000: if er then print " Q " ;:goto170
MB  190 pl = l: ph = h
OP  200 input " start of colour map[7 spaces]
        cc00[7 lefts] " ;h$
MG  210 gosub 1000: if er then print " Q " ;:goto170
JP  220 cl = l: ch = h
LK  230 input " q filename of animation station file " ;f$
CD  240 open1,8,12, " 0:pi. " + f$ + " ,p,r "
GA  250 gosub 2000: rem check for disk error
HM  260 get#1,a$,b$
DC  270 open2,8,11, " @0: " + f$ + " .pic,p,w "
PH  280 print#2,chr$(pl)chr$(ph);
OC  290 gosub 2000: rem check for disk error
KC  300 fori = 1to8000:get#1,a$:print#2,left$(a$ + z$,1);
        :next
BM  310 fori = 1to192:get#1,a$:next: rem get extra
        useless bytes
GB  320 close2
IH  330 open2,8,11, " @0: " + f$ + " .c1,p,w "
AG  340 print#2,chr$(cl)chr$(ch);
HE  350 fori = 1to1000:get#1,a$:print#2,left$(a$ + z$,1);
        :next
JB  360 fori = 1to24:get#1,a$:next
IE  370 close2
MK  380 open2,8,11, " @0: " + f$ + " .c2,p,w "
AP  390 print#2,chr$(0)chr$(216);: rem colour nybbles
JH  400 fori = 1to1000:get#1,a$:print#2,left$(a$ + z$,1);
        :next
LE  410 fori = 1to24:get#1,a$:next
KH  420 close2
FE  430 print " qq The background colour is: "
JA  440 get#1,a$: printasc(a$ + z$)
OB  450 close1: close15
MM  460 end
KE  470 :
AH  1000 rem convert hex f$ to dec h,l
FG  1005 er = 0
JK  1010 if len(h$)<>4then er = 1:return
IK  1020 d = 0:fori = 1to4:h = asc(mid$(h$,i))-48
        :d = d*16 + h + 7*(h>9):next
NO  1030 h = int(d)/256: l = d-h*256
MC  1040 return
OI  1050 :
FE  2000 rem* get disk status subroutine *
FA  2010 input#15,a$,b$,c$,d$
JN  2020 if val(a$)then print " qq disk error:  " a$ " ,
            " b$ " , " c$ " , " d$: end
KA  2030 return
```

Listing 6: The assembly-code source listing for the version 1 compressor.

```
BD  100 sys700   ;enable pal 64
EO  110 ;
DO  120 ; picture compressor -
OJ  130 ; optimizes hi-res picture
NK  140 ; and saves on disk
IC  150 ; this version converts file#8 (r)
AK  160 ; to file#9 (w) with same load addr
DN  170 ; sys(*)  compresses 8 to 9
JC  180 ; sys(* + 3) loads 8 to memory
ED  190 ;
MF  200 ;save " @0:comp1.pal " ,8
IE  210 ;
EE  220 .opt oo
HN  230       *       =       $c000
GG  240 ;
ON  250               jmp   compress
PH  260               jmp   decomp
EI  270 ;
OC  280 repcount  .byte 1
IN  290 newbyt    .byte 0
NL  300 prevbyt   .byte 0
KG  310 st8       .byte 0
GL  320 ;
NP  330 picptr        =       $fb
FC  340 ;kernel routines:
FI  350 chrout        =       $ffd2
OE  360 getin         =       $ffe4
PL  370 close         =       $ffc3
KI  380 chkout        =       $ffc9
OA  390 chkin         =       $ffc6
MH  400 clrchn        =       $ffcc
FI  410 readst        =       $ffb7
KB  420 ;
PB  430 compress  =       *
IN  440               jsr     setin
PC  450               jsr     getin           ;start addr lo
CB  460               jsr     setout
NK  470               jsr     chrout
AA  480               jsr     setin
ND  490               jsr     getin           ;start addr hi
KD  500               jsr     setout
FN  510               jsr     chrout
JB  520               jsr     sendpic         ;send picture to file
PI  530               jmp     fin
CJ  540 ;
GN  550 decomp    =       *
AF  560               jsr     setin
HO  570               jsr     getin: sta picptr       ;load addr lo
CL  580               jsr     getin: sta picptr + 1   ;  "     "  hi
JJ  590               ldy     #0
NF  600               jsr     getpic          ;get picture
IN  610 ;
FB  620 fin           =       *
NH  630               jsr     clrchn
IC  640               lda     #8: jsr close
ED  650               lda     #9: jsr close
AI  660               rts
EB  670 ;
OB  680 ;
NK  690 sendpic   =       *
MN  700               jsr     setin
```

```
IL 710              jsr   getin
BI 720              sta   prevbyt
LN 730 nextout      =     *
GI 740              jsr   outbyte
JP 750              lda   st8
OP 760              beq   nextout
OP 770              jsr   writerep      ;last sequence
IP 780              rts
MI 790 ;
GJ 800 ;
FC 810 outbyte      =     *
EF 820              jsr   setin
AD 830              jsr   getin
EM 840              pha
AB 850              jsr   readst: sta st8  ;save status
EO 860              pla
NI 870              sta   newbyt
IC 880              cmp   prevbyt
IC 890              bne   diff
KP 900 ;
BC 910              inc   repcount
MA 920              bne   ok            ;count past 255"?
AB 930              dec   repcount
CK 940              jsr   writerep      ;write rep code
JK 950 ok           =     *
CJ 960              jmp   obfin
AE 970 ;
CL 980 diff         =     *             ;new byte different
BF 990              lda   repcount
EC 1000               cmp  #6
ME 1010               bcs  docode       ;more than 4 the same"?
JG 1020 ;no, just print byte n times
CO 1030               tax               ;# reps for loop
DI 1040               lda  prevbyt
DD 1050               cmp  #254         ;ctrl byte"?
CB 1060               beq  docode       ;yes, have to code it
EK 1070 ;
OH 1080               jsr  setout
FL 1090               lda  prevbyt
GB 1100 nlp           =    *
NC 1110               jsr  chrout
HB 1120               dex
CM 1130               bne  nlp
IE 1140               lda  #1: sta repcount
AF 1150               jmp  obfin
OP 1160 ;
NC 1170 docode        =    *
PI 1180               jsr  writerep
MB 1190 ;
HN 1200 obfin         =    *
DK 1210               lda  newbyt
FH 1220               sta  prevbyt
KL 1230               rts
OE 1240 ;
IF 1250 ;
DN 1260 writerep      =    *            ;write repeat code
MD 1270               jsr  setout
KG 1280               lda  #254        ;special control byte
BO 1290               jsr  chrout
BJ 1300               lda  prevbyt     ;byte to repeat
FP 1310               jsr  chrout
PO 1320               lda  repcount    ;number of reps
JA 1330               jsr  chrout
JC 1340               lda  #1
```

```
KA 1350               sta  repcount    ;re-initialize count
MD 1360               rts
AN 1370 ;
KN 1380 ;
BE 1390 getpic        =    *           ;uncompress
KG 1400               jsr  getin
BO 1410               cmp  #254        ;rep indicator
HH 1420               beq  getrep
EM 1430 ;normal byte, just store it
PA 1440               jsr  storbyt
AJ 1450               jmp  gpfin
KC 1460 ;
HL 1470 getrep        =    *           ;repeat byte n times
EM 1480               jsr  getin       ;byte to repeat
OE 1490               pha
MP 1500               jsr  getin       ;# of repetitions
BL 1510               tax
IH 1520               pla
PF 1530 replp         =    *
GC 1540               jsr  storbyt     ;stick it in memory
FM 1550               dex
DH 1560               bne  replp       ;do it .x times
IJ 1570 ;
NG 1580 gpfin         =    *
AL 1590               jsr  readst      ;check disk status
GG 1600               beq  getpic      ;do until end-of-file
GD 1610               rts
KM 1620 ;
EN 1630 ;
EE 1640 storbyt       =    *           ;put .a in memory
HD 1650               sta  (picptr),y
KL 1660               inc  picptr      ;increment pointer
PG 1670               bne  sb0
NL 1680               inc  picptr + 1
AN 1690 sb0           =    *
AJ 1700               rts
EC 1710 ;
OC 1720 ;
EB 1730 setin         =    *           ;set input to file #8
MO 1740               pha:txa:pha
NC 1750               ldx  #8
LN 1760               jsr  chkin
JD 1770               pla:tax:pla
AO 1780               rts
EH 1790 ;
OH 1800 ;
FI 1810 setout        =    *           ;set output to file #9
MD 1820               pha:txa:pha
PH 1830               ldx  #9
CP 1840               jsr  chkout
JI 1850               pla:tax:pla
AD 1860               rts
KC 1870 .end
```

Listing 7: Memory-based version of The Compressor

```
NH  100 sys 700 ;activate pal 64 assembler
IN  110 ; picture compressor
MI  120 ; optimizes hi-res pic
DK  130 ; and saves on disk
KC  140 ; this version saves from memory
LJ  150 ; at $2000:
JE  160 ; sys(*), " d:filename "
OH  170 ; or loads to load addr:
IE  180 ; sys(* + 3), " filename "
ED  190 ;
AG  200 ; save " @0:comp2.pal " ,8
IE  210 ;
EE  220 .opt oo
HN  230         *       =       $c000
GG  240 ;
ON  250                 jmp     compress
PH  260                 jmp     decomp
EI  270 ;
JJ  280 picture     .word $2000         ;bitmap loc'n
DN  290 repcount    .byte 1             ;counts repetition
BP  300 newbyt      .byte 0             ;current mem byte
EI  310 prevbyt     .byte 0             ;previous byte
DI  320 sendflag    .byte 0             ;comp/decomp flag
MH  330 banksav     .byte 0             ;orig loc 1 value
HN  340 endpic      .word 0             ;end of bitmap
EN  350 ;
KA  360 piclen      =       8000        ;bitmap byte length
FC  370 picptr      =       $fb
NE  380 ;kernel routines:
PK  390 setlfs      =       $ffba
CK  400 setnam      =       $ffbd
JN  410 open        =       $ffc0
LM  420 chrout      =       $ffd2
EJ  430 getin       =       $ffe4
FA  440 close       =       $ffc3
AN  450 chkout      =       $ffc9
EF  460 chkin       =       $ffc6
CM  470 clrchn      =       $ffcc
LM  480 readst      =       $ffb7
AG  490 ;
FG  500 compress    =       *
LO  510                 lda     #1
CN  520                 sta     sendflag
NK  530                 ldy     #1          ;secondary address
EB  540                 bne     cp1
MJ  550 ;
AO  560 decomp      =       *
FC  570                 lda     #0
OA  580                 sta     sendflag
LO  590                 ldy     #2          ;secondary address
MJ  600                 cp1     =           *
IN  610 ;
GN  620                 lda     1           ;bank select reg
DH  630                 sta     banksav     ;store for later
IL  640                 lda     #8          ;file #8
BK  650                 tax                 ;device #8
KJ  660                 jsr     setlfs      ;open 8,8,1 or 2
AJ  670                 jsr     $aefd       ;check for comma
MG  680                 jsr     $ad9e       ;evaluate expression
PB  690                 jsr     $ad8f       ;check for string
HA  700                 ldy     #0
CM  710                 lda     ($64),y     ;string length
GG  720                 pha: iny
PL  730                 lda     ($64),y     ;string addr low
JM  740                 tax: iny
IF  750                 lda     ($64),y     ;string addr hi
JI  760                 tay: pla
OC  770                 jsr     setnam      ;filename = above string
LJ  780                 jsr     open        ;open file
FA  790                 ldx     #8          ;file #8 for chkin/out
GJ  800 ;
FH  810                 ldy     #0
FN  820                 lda     sendflag    ;compress or load
IO  830                 beq     nosnd       ;nosnd = load
AK  840                 jsr     chkout      ;output to file
NH  850                 lda     picture
DB  860                 jsr     chrout      ;start addr lo
KG  870                 lda     picture + 1
HB  880                 jsr     chrout      ;start addr hi
LI  890                 jsr     sendpic     ;send picture to file
AO  900                 jmp     ss1         ;close files and exit
PP  910 nosnd       =       *
DI  920                 jsr     chkin       ;get load addr first
EJ  930                 jsr     getin
JP  940                 sta     picptr      ;load addr lo
IK  950                 jsr     getin
DC  960                 sta     picptr + 1  ;load addr hi
PM  970                 jsr     getpic      ;get picture
BE  980 ss1         =       *
EF  990 ;
JA  1000                jsr     clrchn      ;clear i/o channels
NO  1010                lda     #8
DF  1020                jsr     close       ;close file #8
MI  1030                rts                 ;all finished!
GI  1040 ;
AJ  1050 ;
CP  1060 sendpic    =       *           ;compress picture
EN  1070                lda     picture     ;start addr lo
BM  1080                sta     picptr
IL  1090                clc
AL  1100                adc     #<piclen    ;find last pic byte
CO  1110                sta     endpic      ;last byte lo
EG  1120                lda     picture + 1
HE  1130                sta     picptr + 1  ;start addr hi
AD  1140                adc     #>piclen
MD  1150                sta     endpic + 1  ;last byte hi
OP  1160 ;
DI  1170                jsr     getbyt      ;read byte from mem
BD  1180                sta     prevbyt     ;initialize prev byte
CK  1190                ldy     #1          ;get 2nd byte next
GC  1200 ;
LL  1210 nextout    =       *
MP  1220                jsr     outbyte     ;fetch byte or group
EE  1230 ;
BE  1240                lda     picptr + 1  ;see if at pic end
CK  1250                cmp     endpic + 1
EP  1260                bne     sp1
BE  1270                lda     picptr
EL  1280                cmp     endpic
OI  1290                bcc     nextout     ;do next byte
AA  1300                rts
CI  1310 sp1        =       *
MK  1320                bcc     nextout     ;do next byte
PO  1330                jsr     writerep    ;write last group
PJ  1340                rts                 ;all bytes done
```

```
ML 1350 ;
GM 1360 ;
PH 1370 outbyte   =    *          ;check next byte
FF 1380           jsr  getbyt     ;read byte from mem
FJ 1390           sta  newbyt
FL 1400           cmp  prevbyt    ;compare to previous
CB 1410           bne  diff       ;different"?
CA 1420 ;
BL 1430           inc  repcount   ;same, inc count
HG 1440           bne  ok         ;>255 repetitions"?
DJ 1450           dec  repcount   ;set to 255
AE 1460           jsr  writerep   ;write repeat code
LH 1470           lda  #1         ;restart count
JH 1480           sta  repcount
FM 1490 ok        =    *
JJ 1500           jmp  obfin      ;finished outbyte
MF 1510 ;
OM 1520 diff      =    *          ;new byte different
FO 1530           lda  repcount   ;check count
HJ 1540           cmp  #4         ;3 or more the same"?
AC 1550           bcs  docode     ;yes, send rep code
FI 1560 ;no, just print byte n times
OP 1570           tax             ;# reps for loop
JK 1580           lda  prevbyt    ;byte to repeat
PE 1590           cmp  #254       ;ctrl byte"?
FG 1600           beq  docode     ;yes, must code it
AM 1610 ;
ED 1620 nlp       =    *          ;repeat loop
BN 1630           lda  prevbyt
AO 1640           jsr  chrout     ;send byte
LI 1650           dex             ;do .x times
EN 1660           bne  nlp
DE 1670           lda  #1         ;restart count
BE 1680           sta  repcount
FA 1690           jmp  obfin      ;finished subrtn
KB 1700 ;
IO 1710 docode    =    *          ;write repeat code
LK 1720           jsr  writerep
PH 1730           lda  #1         ;restart count
NH 1740           sta  repcount
ME 1750 ;
HA 1760 obfin     =    *
DN 1770           lda  newbyt
CK 1780           sta  prevbyt    ;prev=new
DO 1790           inc  picptr     ;next address
AP 1800           bne  ob1
PD 1810           inc  picptr+1
OE 1820 ob1       =    *
CB 1830           rts
GK 1840 ;
AL 1850 ;
LC 1860 writerep  =    *          ;write repeat code
IL 1870           lda  #254       ;special control byte
PC 1880           jsr  chrout
PN 1890           lda  prevbyt    ;byte to repeat
DE 1900           jsr  chrout
ND 1910           lda  repcount   ;number of reps
HF 1920           jsr  chrout
GH 1930           rts
KA 1940 ;
EB 1950 ;
AN 1960 getbyt    =    *
MK 1970           sei             ;disable interrupts
DN 1980           lda  1          ;cpu bank register
NI 1990           and  #$fc       ;select ram
HK 2000           sta  1
CD 2010           lda  (picptr),y ;read byte
AG 2020           pha
GB 2030           lda  banksav    ;get original state
CI 2040           sta  1          ;and restore
AJ 2050           cli
EJ 2060           pla
CA 2070           rts
GJ 2080 ;
AK 2090 ;
HA 2100 getpic    =    *          ;uncompress
AD 2110           jsr  getin
HK 2120           cmp  #254       ;rep indicator
ND 2130           beq  getrep
KI 2140 ;normal byte, just store it
LC 2150           sta  (picptr),y
FF 2160           inc  picptr     ;next address
HH 2170           bne  gr0
BL 2180           inc  picptr+1
MN 2190 gr0       =    *
OH 2200           jmp  gpfin
IB 2210 ;
AL 2220 getrep    =    *
CL 2230           jsr  getin      ;byte to repeat
MD 2240           pha
KO 2250           jsr  getin      ;# of repetitions
PJ 2260           tax
GG 2270           pla
CE 2280 replp     =    *          ;repeat byte n times
HL 2290           sta  (picptr),y
BO 2300           inc  picptr     ;next address
GA 2310           bne  gr1
ND 2320           inc  picptr+1
MG 2330 gr1       =    *
LN 2340           dex
MP 2350           bne  replp
OK 2360 ;
DI 2370 gpfin     =    *
EF 2380           jsr  readst     ;read disk status
MH 2390           beq  getpic     ;do until end-of-file
ME 2400           rts
GE 2410 .end
```

Indestructible Variables

Tom Hall
Zephyr, Ontario

Remember "SuperNumbers"? Well, this is different. SuperNumbers was published in Volume 6, Issue 01. It presented a program that created a whole new type of variable that was impervious to NEW, CLR, or anything short of tampering with their memory space. The following protects the regular everyday variables we're all familiar with. How many of you have written a program, RUN it, and noticed something you want to change. Enter an age old dilemma: "If I stop and edit that, I lose all my variables, but if I don't make the change my program will continue incorrectly. Hmmm." At this time I would like to say Thank You Tom Hall for eliminating one more classic struggle. - Karl J.H. Hildon

Wouldn't it be nice if all variables, including arrays, were indestructible. Well, here is my solution – a utility program which does just that. To fully understand how it is possible to keep your variables from being destroyed one must first look at how they are stored and how BASIC inserts, changes, or deletes a line when you press return.

The variables are kept in line by a series of pointers extending from $2d–$34. The address at ($2d) points to the end of your program and to the start of variable storage. The end–of–variables/start–of–arrays is pointed to by ($2f), the end–of–arrays by ($31) and the bottom of string storage is determined by ($33)

When BASIC senses that you have pressed RETURN it checks to see if whatever was on the line started with a number. If it does then BASIC clears all variables (by making their pointers equal to the end–of–program pointer), searches for and deletes the line indicated by the line number, tokenizes the new line, and inserts the new line into your program if anything was after the line number.

Now hold on a second, what if we intercept BASIC when it goes to change a line and somehow make it move the variables up and down at the same time as it is shifting parts of your program around. Well that is almost exactly what I did.

The system vector at ($0302), called WARMSTART, normally points to a routine in the ROMs which decides whether you want to change a line or execute a direct mode command such as RUN. By changing this vector to point at my program it can decide if it is necessary to shift the variables with the program. I use a flag that I will call "variables shifted flag" to signal when the variables are to be moved around. What specifically happens is the following.

My program waits for you press RETURN and then checks for a line number. If it does not find one then a direct mode command (or RETURN on a blank line) is assumed and control is passed back to the ROMs. Otherwise, assuming the "variables shifted flag" was not set, the following occurs. All of those pointers previously mentioned are put away in temporary storage. Then the end–of–program pointer is moved up to equal the end–of–arrays pointer. Now when BASIC moves parts of your program around the variables will all float up and down with it. The "variables shifted flag" is now set and control is given back to the ROMs.

The next time BASIC is ready for a line it again vectors through to my program. This time however the "variables shifted flag" is set and the variable pointers must be recovered before anything is allowed to happen. This is done in the following manner.

The "variables shifted flag" is cleared and the previously stored end–of–program pointer is subtracted from the present end–of–program pointer. This gives us the difference between the old position of the variables and their new position. This difference (it may be positive or negative) is then added to each of the stored pointers (except for the string pointer) in turn and then they are put back into their usual locations at $2d–$32. The string pointer is simply replaced with its old value since it was not involved in the shifting but it was made equal to the top of memory pointer by BASIC. Now everything is as it was before you modified your program. All of your variables, strings and arrays are intact.

You may restart your program by a GOTO to a convenient line number. Be forewarned however that CONT will not work (you will only get a ?CAN'T CONTINUE ERROR) and RUN or CLR will still clear all variables.

The utility does not provide any increase in speed of your program, however it will give you the edge when programming. You can repeatedly test parts such as output subroutines without having to run the rest of your program over again and wasting your valuable time.

One more thing: the program points the RESTORE vector ($0318) to a part of itself so that if you press RUN/STOP–RESTORE any machine language program gone crazy will still be stopped, but the system vectors are not changed to point to the ROMs. This means that the utility can not be accidentally disconnected so your variables can not be inadvertently lost.

The only drawback I have found when using my utility is that when used with another utility such as POWER (it uses the same WARMSTART vector) they tend to cancel each other out. Simply typing FIX brings POWER back, but disables my program. Type SYS 49152 to re–enable your indestructible variables.

To start using the utility, type in the program in Listing 1. When RUN, this program generates a machine code program on disk that you load like this:

load " vars–indestruct " ,8,1

That ' ,1 ' is very important – it specifies a non-relocating LOAD to $C000 where the machine code lives. Once you have LOADed it type NEW, then SYS 49152 and all your variables are indestructible.

For those of you interested in studying the technique in greater depth I have included the source code in Listing 2.

Listing 1: Create " VARS–INDESTRUCT "

```
LA  100 rem create program " vars–indestruct "
GI  110 printchr$(147) " creating disk file "
GL  120 rem on disk
PG  130 open1,8,3, " vars–indestruct,p,w "
NA  140 print#1,chr$(0)chr$(192);
BP  150 ch = 0 : for i = 49152to49416
PL  160 read d : ch = ch + d
PN  170 print#1,chr$(d);
AA  180 next i : print " checksum = " ;ch
LG  190 close1 : print " should be 29338 "
CO  200 print " q program is now on disk "
CN  210 end
GD  220 data 169,  27, 141,  24,   3, 169, 192, 141
LB  230 data  25,   3, 169,  52, 141,   2,   3, 169
GD  240 data 192, 141,   3,   3, 169, 209, 160, 192
PP  250 data  76,  30, 171,  72, 138,  72, 152,  72
IG  260 data  32, 225, 255, 208,   9,  32, 132, 255
MF  270 data  32, 129, 255, 108,   2,   3, 104, 168
BD  280 data 104, 170, 104,  64, 173, 208, 192, 240
GH  290 data  75, 169,   0, 141, 208, 192,  56, 165
IL  300 data  45, 237, 204, 192, 133, 251, 165,  46
ED  310 data 237, 205, 192, 133, 252,  24, 173, 200
EF  320 data 192, 101, 251, 133,  45, 173, 201, 192
DF  330 data 101, 252, 133,  46,  24, 173, 202, 192
JF  340 data 101, 251, 133,  47, 173, 203, 192, 101
NG  350 data 252, 133,  48,  24, 173, 204, 192, 101
IJ  360 data 251, 133,  49, 173, 205, 192, 101, 252
NL  370 data 133,  50, 173, 206, 192, 133,  51, 173
BA  380 data 207, 192, 133,  52,  32,  96, 165, 134
BO  390 data 122, 132, 123,  32, 115,   0, 170, 240
GP  400 data 163, 162, 255, 134,  58, 144,   6,  32
MB  410 data 121, 165,  76, 225, 167,   8,  72, 138
AN  420 data  72, 152,  72, 169, 128, 141, 208, 192
IK  430 data 160,   7, 185,  45,   0, 153, 200, 192
AH  440 data 136,  16, 247, 165,  49, 133,  45, 133
GO  450 data  47, 165,  50, 133,  46, 133,  48, 104
AP  460 data 168, 104, 170, 104,  40,  76, 156, 164
EK  470 data   0,   0,   0,   0,   0,   0,   0,   0
HP  480 data   0,  13,  10,  73,  78,  68,  69,  83
KG  490 data  84,  82,  85,  67,  84,  73,  66,  76
KG  500 data  69,  32,  86,  65,  82,  73,  65,  66
DH  510 data  76,  69,  83,  32,  45,  32,  66,  89
GG  520 data  32,  84,  79,  77,  32,  72,  65,  76
JG  530 data  76,  13,  13,  10,  18,  65,  67,  84
GJ  540 data  73,  86,  65,  84,  69,  68,  13,  10
ML  550 data   0
```

Listing 2: VARS–INDESTRUCT Source Code

```
JL  1000 sys700
KF  1010 .opt oo
MD  1020 * = 49152
MH  1030 ;
OF  1040 ;truly indestructible variables
AJ  1050 ;
NA  1060 ;(c)june 11, 1985 by tom hall
EK  1070 ;
LP  1080 ;equates
IL  1090 ;
KK  1100 vars          =    45          ;beginning of variables
FH  1110 arrays        =    47          ;end of var/begin of
                                        arrays
LK  1120 endarrays     =    49          ;end of arrays
EI  1130 strings       =    51          ;beginning of string
                                        storage
PD  1140 temp          =    251         ;temporary storage
BJ  1150 restore       =    792         ;restore vector
DI  1160 warmstart     =    770         ;warmstart vector
HP  1170 chkstop       =    $ffe1
NG  1180 initio        =    $ff84
PK  1190 initvid       =    $ff81
MH  1200 rewarm        =    $a49c
BI  1210 printmes      =    $ab1e
KD  1220 ;
ED  1230 start         =    *
OE  1240 ;
IP  1250 ;insert new vectors
CG  1260 ;
HN  1270               lda  #<dorestore
CD  1280               sta  restore
HO  1290               lda  #>dorestore
NC  1300               sta  restore + 1
EJ  1310 ;
EC  1320               lda  #<dowarm
FD  1330               sta  warmstart
ED  1340               lda  #>dowarm
AD  1350               sta  warmstart + 1
MJ  1360 print sign on message
DG  1370               lda  #<message
JM  1380               ldy  #>message
HH  1390               jmp  printmes
OO  1400 ;
NN  1410 dorestore     =    *
CA  1420 ;
DI  1430               pha             ;save registers
EF  1440               txa
GC  1450               pha
LG  1460               tya
KD  1470               pha
OG  1480               jsr  chkstop    ;is stop key down
AP  1490               bne  nostop
IK  1500               jsr  initio     ;yes
PH  1510               jsr  initvid
EP  1520               jmp  (warmstart)
FA  1530 nostop        =    *          ;no
MI  1540               pla
NN  1550               tay
AK  1560               pla
```

```
NO 1570              tax
EL 1580              pla
KP 1590              rti
GL 1600 ;
LF 1610 dowarm   =   *
KM 1620 ;
DI 1630              lda  flag          ;flag starts as zero
HD 1640              beq  noshift
GD 1650 fix variable pointers
MP 1660              lda  #0            ;kill the flag
JO 1670              sta  flag
GA 1680 ;
DL 1690 find correction amount for pointers
FC 1700              sec
AD 1710              lda  vars
EG 1720              sbc  storage+4
NG 1730              sta  temp
KF 1740              lda  vars+1
GI 1750              sbc  storage+5
HJ 1760              sta  temp+1
AG 1770 ;
GN 1780              clc                ;correct stored
                                        pointers
EE 1790              lda  storage       ;fix start of vars
HG 1800              adc  temp
CN 1810              sta  vars
MK 1820              lda  storage+1
BJ 1830              adc  temp+1
MP 1840              sta  vars+1
KM 1850              clc                ;fix end of vars/start
                                        of arrays
IN 1860              lda  storage+2
NK 1870              adc  temp
MN 1880              sta  arrays
KP 1890              lda  storage+3
HN 1900              adc  temp+1
ML 1910              sta  arrays+1
MF 1920              clc                ;fix end of arrays
GC 1930              lda  storage+4
DP 1940              adc  temp
AH 1950              sta  endarrays
IE 1960              lda  storage+5
NB 1970              adc  temp+1
HG 1980              sta  endarrays+1
AG 1990 restore bottom of strings
EH 2000              lda  storage+6
KG 2010              sta  strings
MI 2020              lda  storage+7
HF 2030              sta  strings+1
OG 2040 ;
DA 2050 noshift      jsr  $a560         ;copied from the
HC 2060              stx  $7a           ;rom routines -
                                        change,
MJ 2070              sty  $7b           ;insert or delete a
                                        line,
KF 2080              jsr  $0073         ;or execute a direct
                                        mode
CI 2090              tax                ;command
CK 2100              beq  dowarm
HF 2110              ldx  #$ff
HJ 2120              stx  $3a
GG 2130              bcc  linenum
AO 2140              jsr  $a579         ;tokenize and
HA 2150              jmp  $a7e1         ;execute command
OE 2160 linenum  =   *                  ;starts with a line
                                        number
OA 2170 save registers
MD 2180              php
KA 2190              pha
ME 2200              txa
OB 2210              pha
DG 2220              tya
CD 2230              pha
GD 2240 ;
GI 2250 prepare for variable shift
NK 2260 and set shift flag
EF 2270 ;
LE 2280              lda  #128
FF 2290              sta  flag
OB 2300              ldy  #7            ;save variable pointers
NE 2310 store1   =   *
AH 2320              lda  vars,y
OA 2330              sta  storage,y
PN 2340              dey
IK 2350              bpl  store1
IF 2360              lda  endarrays     ;set start of variables
DI 2370              sta  vars          ;and start of arrays to
CJ 2380              sta  arrays        ;end of arrays
DM 2390              lda  endarrays+1
MC 2400              sta  vars+1
AL 2410              sta  arrays+1
DM 2420 recover registers
GA 2430              pla
HF 2440              tay
KB 2450              pla
HG 2460              tax
OC 2470              pla
EH 2480              plp
ID 2490              jmp  rewarm        ;re-enter warmstart
                                        routine
KD 2500 ;
KL 2510 storage  =   *
BL 2520 .word 0,0,0,0
FA 2530 flag     =   *
GC 2540 .byt 0
MG 2550 ;
NH 2560 message  =   *
AI 2570 ;
KC 2580 .byt 13,10
GM 2590 .asc " indestructible variables - by tom hall "
ML 2600 .byt 13,13,10
AF 2610 .asc " reactivated "
GJ 2620 .byt 13,10,0
```

Disk Un–Assembler For The Commodore 64

J. Lothian
Ottawa, Ont.

Create Real Source From Object Files

As programmers, we frequently acquire machine language code programs that we would like to analyze and understand, but the lack of an assembler source code file or listing severely limits our investigations. Perhaps we wish to understand the programmer's technique, to identify the usable subroutines, or to reconstruct a program that we previously wrote but for which the documentation was lost. We may also wish to modify the program slightly to relocate it, or to modify zero page storage, or to change the I/O in some fashion. What we require in such instances is a utility program that will scan a given machine code program and produce a corresponding assembly language source file that is usable by our assembler language development package.

There are utilities available within the monitors of most assembler packages that satisfy some of these requirements and they are referred to as dissemblers. Unfortunately, the dissemblers provided with these packages have two serious shortcomings. First, dissemblers typically display the assembled code on the screen or printer but do not create source files that are compatible with the programmer's assembler/editor package. Second, the dissemblers do not provide listings with symbolic labels.

Symbolic assemblers use labels to denote locations (addesses) and expressions (values). Labels can be attached to any instruction or expression to denote the memory location and then all jumps or branches within the code are done by referencing these labels. The labels can be any combination of letters and numbers, but must start with a letter. The benefits of labels are that they make code automatically relocatable and they reduce the burden on the programmer since all references to locations are relative and defined by a comprehensible mnemonic/label. Also with labels, branching operations do not involve complex hexadecimal calculations.

In the September 1982 volume of the Transactor, Paul Higginbottom provided such a utility for CBM BASIC 4.0 machines and called it an "Un–Assembler" in order to distinguish it from a disassembler. That version was not appropriate for the C64 or VIC20. This version will work with any Commodore machine including the C64 and the VIC20 and it includes several extra features. (For machines other than the C64 all POKEs and the special characters in the print statements should be removed.)

A program that requires un–assembly does not need to be resident in core since the utility reads the machine language code from disk and writes the source code on the same disk (using the 1541) or on another disk if a dual disk drive is available. The source code files are compatible with the Commodore 64 Macro Assembler Development System, but the program can be easily modified to accommodate any other assembler format. The output from the Un–assembler can also be directed to the screen or printer instead of the disk drive.

Address labels are generated by the utility but not expression labels. Label locations within the program and outside the program are generated by the utility. Addresses outside the program plus addresses that occur in the middle of instructions are symbolically defined through the EQUATE(=) directive. Labels are defined as character strings starting with "AD" followed by the address of the target location in hexadecimal notation.

The utility allows the programmer to choose the starting location within the program for the un–assembly. This will allow un–assembly of subroutines and the avoidance of byte tables. The utility does not convert BIT operations to BYTE operations as Paul Higginbottom's version did. I have rarely encountered problems with this instruction and I have found more use for the different start locations. It is easy to install this feature by inserting the line:

```
61  MD(36)=0: MD(44)=0
```

The Un–assembler makes four passes through the machine code file. In the first pass, the starting and ending addresses of the machine code program are obtained. This allows the utility to distinguish between in–range jumps and branches and those with targets outside the program. These addresses are printed to provide information to the programmer.

In the second pass, the user is asked to provide the starting address for the un–assembly. Starting from this location, a label table is constructed for all the jump and branch targets. The program treats in–range and out–of–range labels differently.

In the third pass, invalid in-range labels are identified. Byte tables or other problems with the code may result in a jump or a branch instruction that has a target in the middle of another instruction. Such labels are invalid and treated by the utility like an out-of-range label.

In the fourth pass, the Un-assembler creates the assembler source files. The files are created in a format that conforms to the Commodore Macro Assembler and they can be modified with the Commodore Editor. At the start of the file all the out-of-range and invalid labels are defined by using the EQUATE(=) assembler directive. Following this comes the assembly code with labels attached to any in-range and valid target lines. The utility will automatically create extra files if the first file gets too large. The output produced can be re-assembled into exactly the same machine code with which you started.

When using the utility, it will occasionally appear to freeze with the disk drive red light remaining on. This is natural and it occurs because the machine is undergoing garbage collection. (This has been discussed in detail in previous issues of the Transactor.)

Unfortunately, the Un-assembler will occasionally produce incomprehensible source code. First, one of the most important features of an assembly language listing is the use of mnemonic labels. Letter combinations are chosen for labels that are suggestive of their significance in the program. Such information can not be reasonably recovered from an unknown machine code program.

Second, there are apt to be minor ambiguities in translating machine code into a given assembly language that are difficult to resolve. For example, is the data byte $4B to be interpreted as $4B, 75, 'A', or %01001011?

Third, there are inherent problems in dissembling machine code programs containing tables of characters, addresses, data bytes, etc. While it is unlikely that the bytes in a table would constitute properly spaced opcodes, it can not be ruled out. Such tables will tend to be improperly identified and this will prevent accurate decoding of surrounding machine code.

Fourth, multiple entry points to subroutines hidden by BIT instructions may reduce the readability of the assembler code.

Editing of the source files produced by the utility can reduce some of these problems. Despite its limitations, the Un-assembler will be a great benefit in analyzing machine code programs for which there is no original source code.

The Un-Assembler

```
OA  10 rem disk un-assembler.long c64
AE  20 rem originally by paul higginbottom
NG  30 rem modified by j. lothian, ottawa, ontario
LC  40 rem initialize variables
BP  50 poke53280,12:poke53281,15:a$=" ":q=.
       :p=.:n=.:n$=" ":p$=" ":de=.:i=.:bc=.
JB  60 n1$=chr$(0):he$="0123456789abcdef"
       :xx$=chr$(13):ps=1:mh=256:lf=1000:ot=.
EG  70 rem
GO  80 print" Sqqq          c-64 disk un-assembler"
JL  90 print" qqq            jack lothian"
       :print" qq            please wait"
CI  100 rem
HA  110 rem arrays defining assembler op codes
MK  120 dim md(255),mn$(255),l1(500),l2(500)
OH  130 fori=1to151:reada$,a,b:mn$(b)=a$
       :md(b)=a:next
KK  140 rem
KD  150 rem check where source should be listed
AJ  160 print" q source code on print (p), screen (s), "
       :input" or disk (d)";ot$
FE  170 if ot$<>"p"andot$<>"s"andot$<>"d"
       then160
CN  180 rem
PD  190 rem get object and source file names
KL  200 input" q drive number of the program(0 or 1)";fd
       :iffd<>0andfd<>1then200
LN  210 input" program filename";f$
NI  220 iflen(f$)>16thenprint" qr error - file name is
       too long q ":goto210
NJ  230 gosub2410:f$=chr$(fd)+":"+f$+",p,r"
FM  240 open1,8,9,f$:gosub2220:close1
NO  250 ifea<>0thenclose15:goto210
IF  260 if ot$<>"d"then330
AJ  270 input" q drive number for the source file";fo
       :iffo<>0andfd<>1then270
HI  280 input" source filename";of$
MM  290 iflen(of$)>12thenprint" qr error - file name is
       too long q ":goto280
NC  300 of$=str$(fo)+":"+of$+".":gosub2420
EF  310 rem
EG  320 rem convert bit to .byte option
FI  330 print" q do you wish bit operations
       converted[4 spcs]to byte operations?"
IL  340 input" q yes (y) or no (n)";an$
LK  350 if an$<>"n" and an$<>"y" then 340
PE  360 if an$="y" then md(36)=14:md(44)=14
AJ  370 rem
LJ  380 rem first pass - find start and end addresses
AD  390 gosub2380:gosub2260:s=asc(a$+n1$)
       +asc(b$+n1$)*mh:e=s
LL  400 get#1,a$:e=e+1:ifst=0then400
IE  410 close1:de=s:hn=3:gosub2180
```

```
OD | 420 print " q starting address is:[1 spc] "
   |     ;s; " ($ " ;h$; " ) "
MP | 430 de = e:gosub2180:print " q ending address
   |     is:[3 spcs] " ;e; " ($ " ;h$; " ) "
PG | 440 print " q length of the file is " ;e-s + 1; " bytes "
AO | 450 rem
PG | 460 rem read start address for un-assembly
KG | 470 print " q un-assembly starting address "
JE | 480 input " [8 spcs]in decimal or hex ($) " ;an$
PL | 490 gosub2480:sa = de:if sa = 0thensa = s
EN | 500 if sa<sorsa = >ethenprint " qr error - out of
   |     range address " :goto480
MB | 510 rem
CN | 520 rem second pass - construct label table
JH | 530 gosub2380:p = s-1:gosub2260
CN | 540 gosub2090:gosub2120:op = q
DA | 550 on n gosub580,590,630,590,590,590,600,
   |     600,600,600,590,590,580,580
LJ | 560 ifp< = ethen540
AF | 570 close1:goto860
AG | 580 return
PK | 590 gosub2090:p = p + 1:return
KM | 600 gosub2090:ad = q:gosub2090:ad = ad + q*mh
CM | 610 p = p + 2:ifad<sorad>ethengosub760:return
GF | 620 gosub680:return
NO | 630 gosub2090:ad = p + q + (q>127)*mh + 2
JI | 640 p = p + 1:ifad<sorad>ethen return
EH | 650 gosub680:return
CL | 660 rem
DC | 670 rem labels for addresses of in range branches,
   |     jumps, lda, etc
OM | 680 lf = 1:fori = 1tolb + 1:t = l1(i):ift = adthenlf = 0
   |     :goto720
CC | 690 if t<adandt<>0 then 720
FC | 700 if t>adandlf then l1(i) = ad:ad = t:goto720
FJ | 710 if t = 0andlf then l1(i) = ad
JE | 720 next:if lf then lb = lb + 1
GP | 730 return
CA | 740 rem
PJ | 750 rem labels for addresses out of range
GG | 760 ifop<>32andop<>76andop<>108thenreturn
KC | 770 if ad = 0thenreturn
EE | 780 lf = 1:fori = 1tolo + 1:t = l2(i):ift = adthenlf = 0
   |     :goto820
HI | 790 if t<adandt<>0 then 820
PI | 800 if t>adandlf then l2(i) = ad:ad = t:goto820
MP | 810 if t = 0andlf then l2(i) = ad
OO | 820 next:if lf then lo = lo + 1
KF | 830 return
GG | 840 rem
CG | 850 rem print summary of label counts
EH | 860 print " q number of in range labels: " ;lb
AJ | 870 print " q number of out of range labels: " ;lo
OI | 880 rem
OI | 890 rem third pass - check if all labels valid
GH | 900 if lb = 0 then 1120
CB | 910 gosub2380:p = s-1:gosub2260:i = 1:ad = l1(i)
EM | 920 gosub2090:gosub2120:ifad = ptheni = i + 1
   |     :ad = l1(i)
CP | 930 on n gosub970,980,980,980,980,980,
   |     1010,1010,1010,1010
GG | 940 if n>10 then on (n-10) gosub980,980,970,970
PC | 950 ifp< = ethen920
KA | 960 close1:goto1090
GO | 970 return
LF | 980 p = p + 1:if p>e then return
FD | 990 if ad = p then gosub1060
HN | 1000 gosub2090:return
LH | 1010 p = p + 2:if p>e then return
FK | 1020 if ad = porad = p-1then gosub1060
EE | 1030 gosub2090:gosub2090:return
OC | 1040 rem
HF | 1050 rem invalid label change to out of range label
KC | 1060 lv = lv + 1:lo = lo + 1:l2(lo) = ad:i = i + 1
   |     :ad = l1(i):return
ME | 1070 rem
KC | 1080 rem print count of invalid addresses
KN | 1090 if lv <>0 then print " q number of invalid
   |     addresses: " lv
KG | 1100 rem
AH | 1110 rem fourth pass - output assembler code
EH | 1120 gosub2380
II | 1130 rem
LL | 1140 rem open source and machine code files
LK | 1150 nf$ = of$ + " 1.s " :p = s-1:nf = 2
GG | 1160 gosub2320:gosub2220:gosub2260
   |     :nf = 2:lc = 1
AL | 1170 rem
ML | 1180 rem write starting address
FH | 1190 de = p + 1:hn = 3:gosub2180:p$ = " [1 spc]* = $ "
   |     + h$ + " ; <starting address> " :gosub2150
OM | 1200 rem
IF | 1210 rem assign label values for addresses out
   |     of range
FP | 1220 if lo = 0 then1270
JJ | 1230 p$ = " ;<out of range jumps and subs> "
   |     :gosub2150
IP | 1240 fori = 1tolo:de = l2(i):gosub2180
AB | 1250 if i = lo-lv + 1 then p$ = " ;[6 spcs]<invalid
   |     labels> " :gosub2150
GK | 1260 p$ = " [5 spcs]ad " + h$ + " = $ " + h$
   |     :gosub2150:next
HL | 1270 lt = 1:t = l1(lt)
NL | 1280 iflc<>lfthen1350
IC | 1290 rem
OL | 1300 rem after 1000 lines create new file
NC | 1310 nf$ = of$ + mid$(str$(nf),2) + " .s " :p$ = " ; "
   |     :gosub2150:p$ = " .fil " + nf$:gosub2150
GG | 1320 gosub2450:nf = nf + 1:lc = 1:gosub2320
   |     :gosub2220
AF | 1330 rem
AK | 1340 rem start reading op codes
```

```
EE | 1350 gosub2090:op=q:gosub2120
OG | 1360 rem
KB | 1370 rem check if this is a labeled address
KG | 1380 pp$="[8 spcs]":if t=0then1410
CE | 1390 if t<p then lt=lt+1:t=l1(lt):goto1380
LK | 1400 if t=p then de=p:hn=3:gosub2180
   |       :pp$="ad"+h$+"[2 spcs]":lt=lt+1:t=l1(lt)
GL | 1410 if n<>0thenpp$=pp$+n$+"[1 spc]"
BB | 1420 on (n+1) gosub1480,1510,1550,1580,
   |       1640,1670,1700,1730,1790,1850,1910
MJ | 1430 if n>10then on (n-10) gosub1950,1980,2010
KI | 1440 if p>=e then 2700
PK | 1450 goto1280
CN | 1460 rem
FL | 1470 rem illegal op code .byte assumed
CB | 1480 de=op:hn=1:gosub2180:p$=pp$
   |       +".byte $"+h$:gosub2150:return
AP | 1490 rem
AA | 1500 rem implied mode
EK | 1510 p$=pp$:gosub2150:return
PN | 1520 gosub2150:return
IB | 1530 rem
KI | 1540 rem immediate mode
IK | 1550 gosub2090:de=q:hn=1:gosub2180:p$=pp$
   |       +"#$"+h$:gosub2150:p=p+1:return
GD | 1560 rem
CM | 1570 rem relative mode (branches)
AK | 1580 gosub2090:ad=p+q+(q>127)*mh+2
   |       :de=ad:hn=3:gosub2180
OE | 1590 ifad<sorad>ethenp$=pp$+"$"+h$
   |       :goto1610
GL | 1600 p$=pp$+"ad"+h$
LK | 1610 gosub2150:p=p+1:return
CH | 1620 rem
EE | 1630 rem zero page mode
IN | 1640 gosub2090:p=p+1:de=q:hn=1:gosub2180
   |       :p$=pp$+"$"+h$:gosub2150:return
AJ | 1650 rem
PL | 1660 rem x-indexed zero page mode
BC | 1670 gosub2090:p=p+1:de=q:hn=1:gosub2180
   |       :p$=pp$+"$"+h$+",x":gosub2150:return
OK | 1680 rem
AO | 1690 rem y-indexed zero page mode
BE | 1700 gosub2090:p=p+1:de=q:hn=1:gosub2180
   |       :p$=pp$+"$"+h$+",y":gosub2150:return
MM | 1710 rem
JJ | 1720 rem absolute mode
PG | 1730 gosub2090:ad=q:gosub2090:ad=ad+q*mh
   |       :p=p+2:de=ad:hn=3:gosub2180
KL | 1740 p$=pp$+"ad"+h$:ifad>=s
   |       andad<=ethen1760
AA | 1750 if op<>32andop<>76andop<>108orad=0
   |       then p$=pp$+"$"+h$
PM | 1760 gosub2150:return
IA | 1770 rem
AP | 1780 rem x-indexed absolute mode
LK | 1790 gosub2090:ad=q:gosub2090:ad=ad+q*mh
   |       :p=p+2:de=ad:hn=3:gosub2180
PF | 1800 p$=pp$+"ad"+h$+",x":ifad>=s
   |       andad<=ethen1820
EA | 1810 if op<>32andop<>76andop<>108orad=0
   |       then p$=pp$+"$"+h$+",x"
LA | 1820 gosub2150:return
EE | 1830 rem
PC | 1840 rem y-indexed absolute mode
HO | 1850 gosub2090:ad=q:gosub2090:ad=ad+q*mh
   |       :p=p+2:de=ad:hn=3:gosub2180
EK | 1860 p$=pp$+"ad"+h$+",y":ifad>=s
   |       andad<=ethen1880
CE | 1870 if op<>32andop<>76andop<>108orad=0
   |       then p$=pp$+"$"+h$+",y"
HE | 1880 gosub2150:return
AI | 1890 rem
JF | 1900 rem indirect mode
DC | 1910 gosub2090:ad=q:gosub2090:ad=ad+q*mh
   |       :p=p+2:de=ad:hn=3:gosub2180
FJ | 1920 p$=pp$+"(ad"+h$+")":gosub2150:return
IK | 1930 rem
PJ | 1940 rem x-indexed indirect mode
LF | 1950 gosub2090:p=p+1:de=q:hn=1:gosub2180
   |       :p$=pp$+"($"+h$+",x)":gosub2150:return
GM | 1960 rem
AM | 1970 rem y-indexed indirect mode
PK | 1980 gosub2090:p=p+1:de=q:hn=1:gosub2180
   |       :p$=pp$+"($"+h$+"),y":gosub2150:return
EO | 1990 rem
JB | 2000 rem accumulator mode
KE | 2010 p$=pp$+"a":gosub2150:return
CA | 2020 rem
DP | 2030 rem bit converted to .byte operation
DL | 2040 de=op:hn=1:gosub2180:bc=bc+1
CL | 2050 p$="[8 spcs]"+".byte $"+h$+";<this was
   |       a bit instruction>"
LP | 2060 gosub2150:return
ED | 2070 rem
PH | 2080 rem read a byte (a$) from file and calculate
   |       ascii value (q)
DD | 2090 get#1,a$:q=asc(a$+n1$)::return
CF | 2100 rem
BB | 2110 rem decode instruction
DI | 2120 p=p+1:n$=mn$(q):n=md(q):return
AH | 2130 rem
LG | 2140 rem output data line for assembler
AN | 2150 p$=p$+xx$:print#6,p$;:gosub2220
   |       :lc=lc+1:return
OI | 2160 rem
FL | 2170 rem decimal (de) to hex (h$) conversion
LJ | 2180 dx=de:h$="":form=hnto0step-1
   |       :n%=dx/(16↑m):dx=dx-n%*16↑m
CO | 2190 h$=h$+mid$(he$,n%+1,1):next:return
GL | 2200 rem
OD | 2210 rem read disk error channel
```

```
NK | 2220 input#15,ea,eb$,ec,ed:if ea then
   |       print''[qr]disk error'';ea;eb$;''[q]''
CN | 2230 return
ON | 2240 rem
MF | 2250 rem open file and get first two bytes
PK | 2260 open1,8,12,f$:get#1,a$,b$
HA | 2270 if ps<=2thenreturn
MK | 2280 if p<(sa-1)thengosub2090:p=p+1
   |       :goto2280
OA | 2290 return
KB | 2300 rem
NA | 2310 rem open source file
FI | 2320 ifot$=''d''thenp$=''@''+nf$+
   |       '',s,w'':open6,8,3,p$:gosub2220
   |       :return
AD | 2330 ifotthenreturn
HO | 2340 ot=1:ifot$=''p''thenopen6,4:return
PA | 2350 open6,3:return
GF | 2360 rem
MO | 2370 rem print pass number
KC | 2380 print''[qr]pass #'';ps;'' of the file to
   |       be decoded'':ps=ps+1:return
EH | 2390 rem
OP | 2400 rem initialize the disk drive
LC | 2410 open15,8,15,''i''+str$(fd):return
HC | 2420 print#15,''i''+str$(fo):return
MJ | 2430 rem
JM | 2440 rem end of source file
MB | 2450 print#6,chr$(0);:close6:return
KL | 2460 rem
HJ | 2470 rem convert string (an$) to decimal
   |       (de) and hex (h$)
FO | 2480 hd=0:l2=0:l3=1:l4=len(an$)
   |       :hn=3
NC | 2490 a1$=mid$(an$,l3,1)
MI | 2500 if a1$<>chr$(32)then2540
LD | 2510 l3=l3+1
LO | 2520 if l3>l4 then de=0:gosub2080
   |       :return
DP | 2530 goto2490
NA | 2540 if a1$=chr$(36) then hd=1
   |       :l3=l3+1
NP | 2550 l2=l4-l3+1:h$=mid$(an$,l3,l2)
PF | 2560 if hd=0 then gosub2640
   |       :gosub2180:return
IC | 2570 rem
GM | 2580 rem hex to decimal (h$ to de)
GE | 2590 de=0:form=1tol2:forw=0to15:if
   |       mid$(h$,m,1)=mid$(he$,w+1,1)
   |       then 2610
NH | 2600 nextw:m=l2:nextm:de=0:return
JI | 2610 de=de+w*(16↑(l2-m)):nextm
   |       :return
KF | 2620 rem
GJ | 2630 rem string (h$) to decimal (de)
BA | 2640 de=0:l2=len(h$):form=1tol2
   |       :forw=0to9
PC | 2650 if mid$(h$,m,1)=mid$(he$,w+1,1)
   |       then 2670
JL | 2660 nextw:m=l2:nextm:de=0:return
NK | 2670 de=de+w*(10↑(l2-m)):nextm:return
GJ | 2680 rem
AD | 2690 rem end of program - close files
CD | 2700 p$=''[1 spc];'':gosub2150:p$=''[2 spcs].end''
   |       :gosub2150:close1:gosub2450:close15
NN | 2710 print''[qr]un-assembly complete'':end
OL | 2720 rem
LK | 2730 rem mnemonic, addressing mode, hex code
GM | 2740 data brk, 1,  0, ora,11,  1, ora, 4,  5, asl, 4,  6
IC | 2750 data php, 1,  8, ora, 2,  9, asl,13, 10, ora, 7, 13
GJ | 2760 data asl, 7, 14, bpl, 3, 16, ora,12, 17, ora, 5, 21
MD | 2770 data asl, 5, 22, clc, 1, 24, ora, 9, 25, ora, 8, 29
DK | 2780 data asl, 8, 30, jsr, 7, 32, and,11, 33, bit, 4, 36
LI | 2790 data and, 4, 37, rol, 4, 38, plp, 1, 40, and, 2, 41
OG | 2800 data rol,13, 42, bit, 7, 44, and, 7, 45, rol, 7, 46
LB | 2810 data bmi, 3, 48, and,12, 49, and, 5, 53, rol, 5, 54
NK | 2820 data sec, 1, 56, and, 9, 57, and, 8, 61, rol, 8, 62
HN | 2830 data rti, 1, 64, eor,11, 65, eor, 4, 69, lsr, 4, 70
MC | 2840 data pha, 1, 72, eor, 2, 73, lsr,13, 74, jmp, 7, 76
HI | 2850 data eor, 7, 77, lsr, 7, 78, bvc, 3, 80, eor,12, 81
GG | 2860 data eor, 5, 85, lsr, 5, 86, cli, 1, 88, eor, 9, 89
OI | 2870 data eor, 8, 93, lsr, 8, 94, rts, 1, 96, adc,11, 97
AA | 2880 data adc, 4,101, ror, 4,102, pla, 1,104, adc, 2,105
IC | 2890 data ror,13,106, jmp,10,108, adc, 7,109, ror, 7,110
GC | 2900 data bvs, 3,112, adc,12,113, adc, 5,117, ror, 5,118
CE | 2910 data sei, 1,120, adc, 9,121, adc, 8,125, ror, 8,126
GO | 2920 data sta,11,129, sty, 4,132, sta, 4,133, stx, 4,134
NN | 2930 data dey, 1,136, txa, 1,138, sty, 7,140, sta, 7,141
JF | 2940 data stx, 7,142, bcc, 3,144, sta,12,145, sty, 5,148
HJ | 2950 data sta, 5,149, stx, 6,150, tya, 1,152, sta, 9,153
ND | 2960 data txs, 1,154, sta, 8,157, ldy, 2,160, lda,11,161
NP | 2970 data ldx, 2,162, ldy, 4,164, lda, 4,165, ldx, 4,166
BP | 2980 data tay, 1,168, lda, 2,169, tax, 1,170, ldy, 7,172
EB | 2990 data lda, 7,173, ldx, 7,174, bcs, 3,176, lda,12,177
NP | 3000 data ldy, 5,180, lda, 5,181, ldx, 6,182, clv, 1,184
FF | 3010 data lda, 9,185, tsx, 1,186, ldy, 8,188, lda, 8,189
FG | 3020 data ldx, 9,190, cpy, 2,192, cmp,11,193, cpy, 4,196
GA | 3030 data cmp, 4,197, dec, 4,198, iny, 1,200, cmp, 2,201
NN | 3040 data dex, 1,202, cpy, 7,204, cmp, 7,205, dec, 7,206
MI | 3050 data bne, 3,208, cmp,12,209, cmp, 5,213, dec, 5,214
MO | 3060 data cld, 1,216, cmp, 9,217, cmp, 8,221, dec, 8,222
CM | 3070 data cpx, 2,224, sbc,11,225, cpx, 4,228, sbc, 4,229
AG | 3080 data inc, 4,230, inx, 1,232, sbc, 2,233, nop, 1,234
DG | 3090 data cpx, 7,236, sbc, 7,237, inc, 7,238, beq, 3,240
ED | 3100 data sbc,12,241, sbc, 5,245, inc, 5,246, sed, 1,248
LC | 3110 data sbc, 9,249, sbc, 8,253, inc, 8, 254
```

Super Sound

Glen Reesor
Camrose, Alberta

SID Sound — The Easy Way!

How would you like to add eighteen sound–related commands to your Commodore 64? Well you can, with Super Sound. Super Sound opens the door to easy and powerful sound manipulation; no more fumbling through all those POKEs. I'll bet you never dreamed of being able to glance at a program and think: Ah yes, that sets the ATTACK to 5, DECAY to 0, SUSTAIN to 12, and RELEASE to 15. Or, how about being able to easily change the SUSTAIN level without changing the RELEASE rate?

Just type in the Super Sound generator program and save it. Before RUNning it, make sure there is a disk in your drive because it will make a copy of Super Sound for you on that disk. Be careful to type in the data statements carefully. Those numbers are the actual Super Sound program. The new program on your disk is your working copy of Super Sound. Just LOAD and RUN it like any basic program.

These eighteen additional commands behave just like normal commands in BASIC — you can use them in program or direct–mode and you can have multiple commands on a line separated by colons. All commands added by Super Sound must be preceded by the back–arrow (top left corner of the keyboard).

Description of Commands

clear	clear SID chip
volume X	set SID volume to X (0–15)
wave X, tri wave X, saw wave X, pulse wave X, noise	set voice X to specified waveform
play X,FREQUENCY	set voice X to specified frequency and start ATTACK cycle
off X	turn off voice X and start RELEASE cycle
attack X,Y	set ATTACK rate for voice X to Y (0–15)
decay X,Y	set DECAY rate for voice X to Y (0–15)
sustain X,Y	set SUSTAIN level for voice X to Y (0–15)
release X,Y	set RELEASE rate for voice X to Y (0–15)
pulse X,Y	set PULSE width for voice X to Y (0–4095)
filter X, low, Y filter X, high, Y filter X, band, Y	FILTER voice X with desired filter, with a cut–off frequency of Y (0–2047) remember FILTER modes are additive, to change the filter, it must first be turned off
filteroff X	turn off FILTER for voice X
sync X	SYNCHRONIZE voice X with another voice, which is determined by the SID chip
syncoff X	turn off SYNCHRONIZATION for voice X
ring X	RING MODULATE voice X with another voice, which is determined by SID
ringoff X	turn off RING MODULATION for voice X
resonance X	set RESONANCE level to X (0–15)
kill	turn on/off voice 3 (toggle)

Fun with SID

Once you have LOADed Super Sound you are ready to create some sounds. Before I wrote this program, I sometimes had trouble getting even one beep out of my 64. To show how easy it is with Super Sound, type the following:

```
←clear
←volume 15
←wave 1,saw
←sustain 1,15
←play 1,2000
```

You should hear a tone coming from your computer. Now, try the other waveforms. All you have to do is type the WAVE command again, but with a different waveform specified. If you want to use PULSE, remember to set the PULSE width.

Now type the command:

```
←off 1
```

The tone should stop. Notice that the tone stopped right away. To change this, type the previous commands, but insert:

```
←release 1,15
```

before the ←play command. Now when you type ←off 1 the tone should die–away very slowly.

Now that you know how easy it is to make sounds, try some experimentation with different waveforms, as well as different values for ATTACK, DECAY, SUSTAIN, and RELEASE. You will be

amazed at what you can command SID to do. Remember to use the ←off 1 command to start the RELEASE cycle; otherwise when you play another note, SID will not start the RELEASE cycle.

Try these four demos that create interesting effects using SYNCHRONIZATION, RING MODULATION, and FILTERING.

```
10 rem synchronization
20 ←clear
30 ←volume 15
40 ←wave 1,saw
50 ←sustain 1,15
60 ←play 1,10000
70 ←sync 1
80 for x=0 to 10000 step 10: ←play 3,x:next x
90 end
```

```
10 rem ring modulation
20 ←clear
30 ←volume 15
40 ←wave 1,tri
50 ←sustain 1,15
60 ←play 1,10000
70 ←ring 1
80 for x=0 to 10000 step 10: ←play 3,x:next x
```

```
10 rem ring modulation and synchronization
20 ←clear
30 ←volume 15
40 ←wave 1,tri
50 ←sustain 1,15
60 ←play 1,10000
70 ←ring 1
80 ←sync 1
90 for x=0 to 10000 step 10: ←play 3,x:next x
```

Try the following program with and without the resonance command.

```
10 rem filtering (with and without resonance)
20 ←clear
30 ←volume 15
40 ←wave 1,saw
50 ←sustain 1,15
60 ←resonance 15
70 ←play 1,3000
80 for x=0 to 2047 step 10: ←filter 1,low,x:next x
```

Now, with the aid of Super Sound, you too can make awesome sound demos on your 64. With a little experimentation you will be making professional–sounding sound effects with ease.

Editor's Note

There is a little less typing ahead than at first appears. Those multiple lines of zeroes (probably .BYTE tables for parameter storage) can be entered quickly by just changing the line number each time. Speaking of BYTE tables, our apologies for omitting the source code for this one – it's just too long. However, this program would be an ideal candidate for the Unassembler program also in this issue. With a little work it wouldn't be hard to convert Super Sound into a TransBASIC module. M.Ed.

Program to create Super Sound PRG file

(In summary, enter it, save it, run it, then load "super sound" and run it.)

```
PA  10 open 8,8,8,"0:super sound,p,w"
KK  20 print#8,chr$(1);chr$(8);
BK  30 read a:ck=ck+a:ifa=256 then50
EH  40 print#8,chr$(a);:goto30
DJ  50 close8:if ck<>257740 then print"error in
        data statements":stop
MD  60 end
JD  70 data  12,   8,  10,   0, 158,  32,  50,  48
JG  80 data  54,  52,   0,   0,   0,   0, 129, 169
DG  90 data  88, 141,   2,   3, 169, 198, 141,   3
FH 100 data   3, 160,   0, 132, 251, 169, 160, 133
OA 110 data 252, 169,  55, 133,   1, 177, 251, 145
AH 120 data 251, 230, 251, 208,   2, 230, 252, 165
LI 130 data 252, 201, 192, 208, 240, 169,  44, 133
EN 140 data 251, 169, 160, 133, 252, 132, 253, 169
HI 150 data   9, 133, 254, 177, 253, 145, 251, 230
GH 160 data 251, 208,   2, 230, 252, 230, 253, 208
DH 170 data   2, 230, 254, 165, 252, 201, 161, 208
NM 180 data 234, 165, 251, 201, 157, 208, 228, 132
EB 190 data 251, 169, 192, 133, 252, 132, 253, 169
LP 200 data  11, 133, 254, 177, 253, 145, 251, 230
IK 210 data 251, 208,   2, 230, 252, 230, 253, 208
AM 220 data   2, 230, 254, 165, 252, 201, 198, 208
BH 230 data 234, 165, 251, 201,  97, 208, 228,  96
OL 240 data   0,   0,   0,   0,   0,   0,   0,   0
IM 250 data   0,   0,   0,   0,   0,   0,   0,   0
CN 260 data   0,   0,   0,   0,   0,   0,   0,   0
MN 270 data   0,   0,   0,   0,   0,   0,   0,   0
GO 280 data   0,   0,   0,   0,   0,   0,   0,   0
AP 290 data   0,   0,   0,   0,   0,   0,   0,   0
KP 300 data   0,   0,   0,   0,   0,   0,   0,   0
EA 310 data   0,   0,   0,   0,   0,   0,   0,   0
OA 320 data   0,   0,   0,   0,   0,   0,   0,   0
IB 330 data   0,   0,   0,   0,   0,   0,   0,   0
CC 340 data   0,   0,   0,   0,   0,   0,   0,   0
MC 350 data   0,   0,   0,   0,   0,   0,   0,   0
GD 360 data   0,   0,   0,   0,   0,   0,   0,   0
AE 370 data   0,   0,   0,   0,   0,   0,   0,   0
DF 380 data   0,   0,   0,   0,   0,   0,   0, 192
EE 390 data 192,  74, 169,  44, 184, 103, 225,  85
JL 400 data 225, 100, 225, 178, 179,  35, 184, 127
GF 410 data 170, 159, 170,  86, 168, 155, 166,  93
LA 420 data 166, 133, 170,  41, 225, 189, 225, 198
MA 430 data 225, 122, 171,  65, 166,  57, 188, 204
BF 440 data 188,  88, 188,  16,   3, 125, 179, 158
LA 450 data 179, 113, 191, 151, 224, 234, 185, 237
BL 460 data 191, 100, 226, 107, 226, 180, 226,  14
LP 470 data 227,  13, 184, 124, 183, 101, 180, 173
FI 480 data 183, 139, 183, 236, 182,   0, 183,  44
NF 490 data 183,  55, 183, 121, 105, 184, 121,  82
EF 500 data 184, 123,  42, 186, 123,  17, 187, 127
CF 510 data 122, 191,  80, 232, 175,  70, 229, 175
ME 520 data 125, 179, 191,  90, 211, 174, 100,  21
BH 530 data 176,  69,  78, 196,  70,  79, 210,  78
FL 540 data  69,  88, 212,  68,  65,  84, 193,  73
AB 550 data  78,  80,  85,  84, 163,  73,  78,  80
LL 560 data  85, 212,  68,  73, 205,  82,  69,  65
NI 570 data 196,  76,  69, 212,  71,  79,  84, 207
AM 580 data  82,  85, 206,  73, 198,  82,  69,  83
```

```
FE  590 data  84,  79,  82, 197,  71,  79,  83,  85
PO  600 data 194,  82,  69,  84,  85,  82, 206,  82
NK  610 data  69, 205, 223,  79, 206,  87,  65,  73
IP  620 data 212,  76,  79,  65, 196,  83,  65,  86
AC  630 data 197,  86,  69,  82,  73,  70, 217,  68
HD  640 data  69, 198,  80,  79,  75, 197,  80,  82
NF  650 data  73,  78,  84, 163,  80,  82,  73,  78
NA  660 data 212,  67,  79,  78, 212,  76,  73,  83
CN  670 data 212,  67,  76, 210,  67,  77, 196,  83
JD  680 data  89, 211,  79,  80,  69, 206,  67,  76
AF  690 data  79,  83, 197,  71,  69, 212,  78,  69
CN  700 data 215,  84,  65,  66, 168,  84, 207,  70
CD  710 data 206,  83,  80,  67, 168,  84,  72,  69
CC  720 data 206,  78,  79, 212,  83,  84,  69, 208
KF  730 data 171, 173, 170, 175, 222,  65,  78, 196
FO  740 data  79, 210, 190, 189, 188,  83,  71, 206
PE  750 data  73,  78, 212,  65,  66, 211,  85,  83
PP  760 data 210,  70,  82, 197,  80,  79, 211,  83
BG  770 data  81, 210,  82,  78, 196,  76,  79, 199
FJ  780 data  69,  88, 208,  67,  79, 211,  83,  73
PB  790 data 206,  84,  65, 206,  65,  84, 206,  80
KJ  800 data  69,  69, 203,  76,  69, 206,  83,  84
MH  810 data  82, 164,  86,  65, 204,  65,  83, 195
IA  820 data  67,  72,  82, 164,  76,  69,  70,  84
IM  830 data 164,  82,  73,  71,  72,  84, 164,  77
PC  840 data  73,  68, 164,  71, 207,   0,   0,   0
AC  850 data   0,   0,   0,   0,   0,   0,   0,   0
KC  860 data   0,   0,   0,   0,   0,   0,   0,   0
ED  870 data   0,   0,   0,   0,   0,   0,   0,   0
OD  880 data   0,   0,   0,   0,   0,   0,   0,   0
IE  890 data   0,   0,   0,   0,   0,   0,   0,   0
CF  900 data   0,   0,   0,   0,   0,   0,   0,   0
MF  910 data   0,   0,   0,   0,   0,   0,   0,   0
GG  920 data   0,   0,   0,   0,   0,   0,   0,   0
AH  930 data   0,   0,   0,   0,   0,   0,   0,   0
KH  940 data   0,   0,   0,   0,   0,   0,   0,   0
EI  950 data   0,   0,   0,   0,   0,   0,   0,   0
OI  960 data   0,   0,   0,   0,   0,   0,   0,   0
IJ  970 data   0,   0,   0,   0,   0,   0,   0,   0
CK  980 data   0,   0,   0,   0,   0,   0,   0,   0
MK  990 data   0,   0,   0,   0,   0,   0,   0,   0
GL 1000 data   0,   0,   0,   0,   0,   0,   0,   0
AM 1010 data   0,   0,   0,   0,   0,   0,   0,   0
KM 1020 data   0,   0,   0,   0,   0,   0,   0,   0
EN 1030 data   0,   0,   0,   0,   0,   0,   0,   0
ON 1040 data   0,   0,   0,   0,   0,   0,   0,   0
IO 1050 data   0,   0,   0,   0,   0,   0,   0,   0
PB 1060 data  67,  76,  69,  65, 210,  86,  79,  76
HO 1070 data  85,  77, 197,  87,  65,  86, 197,  80
MB 1080 data  85,  76,  83, 197,  65,  84,  84,  65
PN 1090 data  67, 203,  68,  69,  67,  65, 217,  83
PC 1100 data  85,  83,  84,  65,  73, 206,  82,  69
FF 1110 data  76,  69,  65,  83, 197,  80,  76,  65
EO 1120 data 217,  79,  70, 198,  70,  73,  76,  84
LF 1130 data  69,  82,  79,  70, 198,  70,  73,  76
HG 1140 data  84,  69, 210,  83,  89,  78,  67,  79
IC 1150 data  70, 198,  83,  89,  78, 195,  82,  73
DC 1160 data  78,  75,  70, 198,  82,  73,  78, 199
LG 1170 data  82,  69,  83,  17,  65,  78,  67, 197
BN 1180 data  75,  73,  76, 204,   0,   0,   0,  84
AB 1190 data  82, 201,  83,  65, 215,  80,  85,  76
KN 1200 data  83, 197,  78,  79,  73,  83, 197,   0
IC 1210 data  76,  79, 215,  66, 175,  72,  73,  71
LE 1220 data 200,   0,   0,  42, 193,  57, 193,  91
ID 1230 data 193, 107, 195,  71, 194, 146, 194, 217
AH 1240 data 194,  36, 195, 208, 193,  28, 194, 144
JG 1250 data 196, 181, 195, 240, 196, 192, 196,  96
EP 1260 data 197,  48, 197, 176, 197, 224, 197,  96
NA 1270 data 169,   0, 141, 253,   3, 170, 160, 255
PC 1280 data 141, 254,   3, 189,  25, 192,  16,   9
JE 1290 data  72, 169,   1, 141, 253,   3, 104,  41
AB 1300 data 127, 200, 232, 209, 122, 208,   8, 173
HC 1310 data 253,   3, 240, 231,  76,   4, 193, 169
LC 1320 data   0, 141, 253,   3, 238, 254,   3, 238
MJ 1330 data 254,   3, 160, 255, 189,  25, 192, 240
JG 1340 data  45,  16,   4, 232,  76, 204, 192, 232
PA 1350 data  76, 245, 192, 200, 230, 122, 208,   2
DH 1360 data 230, 123, 136, 208, 247, 169, 156, 141
OO 1370 data  37, 193, 169, 192, 141,  38, 193, 174
GG 1380 data 254,   3, 240,   7, 238,  37, 193, 202
CM 1390 data  76,  27, 193, 108, 156, 192,  76,  96
LL 1400 data 196, 169,   0, 168, 153,   0, 192, 153
NG 1410 data   0, 212, 200, 192,  25, 208, 245,  96
DL 1420 data  32, 158, 173,  32, 170, 177, 170, 240
CA 1430 data   3,  76,  72, 178, 192,  16,  16, 249
IK 1440 data 140, 252,   3, 173,  24, 192,  41, 240
BK 1450 data  13, 252,   3, 141,  24, 192, 141,  24
HD 1460 data 212,  96,  32, 158, 173,  32, 170, 177
FA 1470 data 170, 240,   3,  76,  72, 178, 152, 240
JN 1480 data 250, 192,   4,  16, 246, 169,   4, 136
FP 1490 data 240,   6,  24, 105,   7,  76, 112, 193
KE 1500 data 141, 252,   3,  32, 253, 174, 169, 128
MB 1510 data 141, 205, 192, 141, 246, 192, 169,  96
LI 1520 data 141,  14, 193,  32, 193, 192, 169,  25
BF 1530 data 141, 205, 192, 141, 246, 192, 169, 169
AN 1540 data 141,  14, 193, 238, 254,   3, 238, 254
PI 1550 data   3, 173, 254,   3,  74, 201,   3, 208
FO 1560 data   5, 169,   4,  76, 181, 193, 201,   4
GA 1570 data 208,   2, 169,   8,  10,  10,  10,  10
BE 1580 data 172, 252,   3, 141, 252,   3, 185,   0
LM 1590 data 192,  41,  15,  13, 252,   3, 153,   0
KE 1600 data 192, 153,   0, 212,  96,   0,   0,  32
FB 1610 data 158, 173,  32, 170, 177, 170, 240,   3
CG 1620 data  76,  72, 178, 152, 240, 250, 192,   4
CM 1630 data  16, 246, 169,   0, 136, 240,   6,  24
AD 1640 data 105,   7,  76, 229, 193, 141, 252,   3
OP 1650 data  32, 253, 174,  32, 158, 173,  32, 112
EJ 1660 data 196,  72, 152, 172, 252,   3, 153,   0
MF 1670 data 192, 153,   0, 212, 104, 200, 153,   0
JF 1680 data 192, 153,   0, 212, 200, 200, 200, 185
EK 1690 data   0, 192,   9,   1, 153,   0, 192, 153
II 1700 data   0, 212,  96,  32, 158, 173,  32, 170
FA 1710 data 177, 170, 240,   3,  76,  72, 178, 152
IL 1720 data 240, 250, 192,   4,  16, 246, 169,   4
DG 1730 data 136, 240,   6,  24, 105,   7,  76,  49
LF 1740 data 194, 168, 185,   0, 192,  41, 254, 153
DP 1750 data   0, 192, 153,   0, 212,  96,  32, 158
KE 1760 data 173,  32, 170, 177, 170, 240,   3,  76
DF 1770 data  72, 178, 152, 240, 250, 192,   4,  16
CO 1780 data 246, 169,   5, 136, 240,   6,  24, 105
BL 1790 data   7,  76,  92, 194, 141, 252,   3,  32
GG 1800 data 253, 174,  32, 158, 173,  32, 170, 177
OD 1810 data 170, 208, 220, 192,  16,  16, 216, 152
GA 1820 data  10,  10,  10,  10, 172, 252,   3, 141
```

```
GL | 1830 data 252,   3, 185,   0, 192,  41,  15,  13
MD | 1840 data 252,   3, 153,   0, 192, 153,   0, 212
IL | 1850 data  96,  32, 158, 173,  32, 170, 177, 170
GI | 1860 data 240,   3,  76,  72, 178, 152, 240, 250
NF | 1870 data 192,   4,  16, 246, 169,   5, 136, 240
FI | 1880 data   6,  24, 105,   7,  76, 167, 194, 141
NH | 1890 data 252,   3,  32, 253, 174,  32, 158, 173
PN | 1900 data  32, 170, 177, 170, 208, 220, 192,  16
OB | 1910 data  16, 216, 152, 172, 252,   3, 141, 252
NJ | 1920 data   3, 185,   0, 192,  41, 240,  13, 252
HP | 1930 data   3, 153,   0, 192, 153,   0, 212,  96
LL | 1940 data  32, 158, 173,  32, 170, 177, 170, 240
DP | 1950 data   3,  76,  72, 178, 152, 240, 250, 192
CF | 1960 data   4,  16, 246, 169,   6, 136, 240,   6
FB | 1970 data  24, 105,   7,  76, 238, 194, 141, 252
MJ | 1980 data   3,  32, 253, 174,  32, 158, 173,  32
AC | 1990 data 170, 177, 170, 208, 220, 192,  16,  16
JJ | 2000 data 216, 152, 172, 252,   3,  10,  10,  10
AA | 2010 data  10, 141, 252,   3, 185,   0, 192,  41
EO | 2020 data  15,  13, 252,   3, 153,   0, 192, 153
CN | 2030 data   0, 212,  96,  32, 158, 173,  32, 170
PE | 2040 data 177, 170, 240,   3,  76,  72, 178, 152
KA | 2050 data 240, 250, 192,   4,  16, 246, 169,   6
KK | 2060 data 136, 240,   6,  24, 105,   7,  76,  57
AD | 2070 data 195, 141, 252,   3,  32, 253, 174,  32
DF | 2080 data 158, 173,  32, 170, 177, 170, 208, 220
ND | 2090 data 192,  16,  16, 216, 152, 172, 252,   3
IC | 2100 data 141, 252,   3, 185,   0, 192,  41, 240
DM | 2110 data  13, 252,   3, 153,   0, 212, 153,   0
IO | 2120 data 192,  96,  32, 158, 173,  32, 170, 177
JJ | 2130 data 170, 240,   3,  76,  72, 178, 152, 240
FG | 2140 data 250, 192,   4,  16, 246, 169,   2, 136
DK | 2150 data 240,   6,  24, 105,   7,  76, 128, 195
NH | 2160 data 141, 252,   3,  32, 253, 174,  32, 158
JK | 2170 data 173,  32, 170, 177, 201,  16,  48,   8
NH | 2180 data 201,  17,  16, 215, 192,   0, 208, 211
HM | 2190 data 170, 152, 172, 252,   3, 153,   0, 192
AI | 2200 data 153,   0, 212, 200, 138, 153,   0, 192
FI | 2210 data 153,   0, 212,  96,  32, 158, 173,  32
JA | 2220 data 170, 177, 170, 240,   3,  76,  72, 178
KE | 2230 data 152, 240, 250, 192,   4,  16, 246, 234
DN | 2240 data 234, 234, 234, 192,   3, 208,   1, 200
KP | 2250 data 140,   2,   0,  32, 253, 174, 169, 145
KA | 2260 data 141, 205, 192, 141, 246, 192, 169,  96
OG | 2270 data 141,  14, 193,  32, 120, 196, 169,  25
PD | 2280 data 141, 205, 192, 141, 246, 192, 169, 169
OL | 2290 data 141,  14, 193, 238, 254,   3, 238, 254
NH | 2300 data   3, 173, 254,   3,  74, 201,   3, 208
LJ | 2310 data   3,  24, 105,   1,  10,  10,  10,  10
GC | 2320 data 141, 254,   3,  32, 243, 197,  32, 158
BE | 2330 data 173,  32, 170, 177, 201,  32,  48,   8
DC | 2340 data 201,  33,  16, 161, 192,   0, 208, 157
OF | 2350 data 170, 152,  41,   7, 141,  21, 192, 141
BH | 2360 data  21, 212, 152,  74,  74,  74, 141, 252
OA | 2370 data   3, 138,  10,  10,  10,  10,  10,  13
GC | 2380 data 252,   3, 141,  22, 192, 141,  22, 212
IJ | 2390 data 234, 234, 234, 234, 173,  23, 192,  13
EH | 2400 data   2,   0, 141,  23, 192, 141,  23, 212
LG | 2410 data 173, 254,   3,  13,  24, 192, 141,  24
IL | 2420 data 192, 141,  24, 212,  96,   0,   0, 169
IN | 2430 data  25, 141, 205, 192, 141, 246, 192, 169
OM | 2440 data 169, 141,  14, 193,  76,  30, 197,  32

CD | 2450 data 247, 183, 164,  20, 165,  21,  96, 160
OJ | 2460 data   1, 177, 122,  16,   4,  41, 127, 145
EL | 2470 data 122,  76, 193, 192,   0,   0,   0,   0
FO | 2480 data   0,   0,   0,   0,   0,   0,   0,  32
FI | 2490 data 158, 173,  32, 170, 177, 170, 240,   3
CN | 2500 data  76,  72, 178, 152, 240, 250, 192,   4
AE | 2510 data  16, 246, 169, 255,  56, 132,   2, 229
HN | 2520 data   2,  45,  23, 192, 141,  23, 192, 141
KN | 2530 data  23, 212,  96,   7,  12,   5,  14,  32
GI | 2540 data  18,   5,   5,  19,  15,  18,   0,  32
BM | 2550 data 158, 173,  32, 170, 177, 170, 240,   3
OA | 2560 data  76,  72, 178, 152, 240, 250, 192,   4
GH | 2570 data  16, 246, 169,   4, 136, 240,   6,  24
ON | 2580 data 105,   7,  76, 213, 196, 168, 185,   0
JD | 2590 data 192,   9,   2, 153,   0, 192, 153,   0
DE | 2600 data 212,  96,   0,   0,   0,   0,   0,  32
NP | 2610 data 158, 173,  32, 170, 177, 170, 240,   3
KE | 2620 data  76,  72, 178, 152, 240, 250, 192,   4
CL | 2630 data  16, 246, 169,   4, 136, 240,   6,  24
CM | 2640 data 105,   7,  76,   5, 197, 168, 169, 255
AI | 2650 data  56, 233,   2,  57,   0, 192, 153,   0
JF | 2660 data 192, 153,   0, 212,  96, 160,   3, 177
DN | 2670 data 122, 201, 203, 208,   7, 169,  75, 145
AE | 2680 data 122,  76, 193, 192,  76, 142, 197,  32
NE | 2690 data 158, 173,  32, 170, 177, 170, 240,   3
KJ | 2700 data  76,  72, 178, 152, 240, 250, 192,   4
CA | 2710 data  16, 246, 169,   4, 136, 240,   6,  24
MP | 2720 data 105,   7,  76,  69, 197, 168, 185,   0
HM | 2730 data 192,   9,   4, 153,   0, 192, 153,   0
PM | 2740 data 212,  96,   0,   0,   0,   0,   0,  32
PN | 2750 data  53, 198,  32, 170, 177, 170, 240,   3
GN | 2760 data  76,  72, 178, 152, 240, 250, 192,   4
OD | 2770 data  16, 246, 169,   4, 136, 240,   6,  24
JF | 2780 data 105,   7,  76, 117, 197, 168, 169, 255
AB | 2790 data  56, 233,   4,  57,   0, 192, 153,   0
FO | 2800 data 192, 153,   0, 212,  96, 160,   3, 177
MB | 2810 data 122, 201, 145, 208,  10, 169,  17, 145
HG | 2820 data 122,  76, 193, 192,  76,   8, 175,  76
CA | 2830 data   8, 175,   0,   0,   0,   0,   0,   0
NE | 2840 data   0,   0,   0,   0,   0,   0,   0,  32
HE | 2850 data  17, 198,  32, 170, 177, 170, 240,   3
CH | 2860 data  76,  72, 178, 192,  16,  16, 249, 152
PB | 2870 data  10,  10,  10,  10, 141, 252,   3, 173
HF | 2880 data  23, 192,  41,  15,  13, 252,   3, 141
JM | 2890 data  23, 192, 141,  23, 212,  96,   0,   0
KC | 2900 data   0,   0,   0,   0,   0,   0,   0, 173
KB | 2910 data  24, 192,  48,   5,   9, 128,  76, 236
GG | 2920 data 197,  10,  74, 141,  24, 192, 141,  24
FL | 2930 data 212,  96, 173, 254,   3, 201,  32, 208
KB | 2940 data  20, 165, 122, 208,   2, 198, 123, 198
HI | 2950 data 122, 169, 175, 160,   0, 145, 122, 230
OM | 2960 data 122, 208,   2, 230, 123,  76, 253, 174
EL | 2970 data 162,   5, 165, 122, 208,   2, 198, 123
GN | 2980 data 198, 122, 202, 208, 245, 169, 145, 160
EI | 2990 data   0, 145, 122, 162,   5, 230, 122, 208
LP | 3000 data   2, 230, 123, 202, 208, 247,  76, 158
NL | 3010 data 173,   0,   0,   0, 162,   3, 165, 122
PL | 3020 data 208,   2, 198, 123, 198, 122, 202, 208
FN | 3030 data 245, 169, 203, 160,   0, 145, 122, 162
OK | 3040 data   3, 230, 122, 208,   2, 230, 123, 202
ID | 3050 data 208, 247,  76, 158, 173,   0,   0, 169
FD | 3060 data  54, 133,   1,  76, 131, 164,   0, 256
```

Eliminating The BASIC Loader

Chris Zamara, Technical Editor

In the pages of **The Transactor** and also in many other computer publications, you'll find machine language programs listed in the form of lots of BASIC DATA statements, and a bit of code to put the DATA values into memory. Our reason for printing the program this way is simply so that any Commodore owner can enter the program, even without a machine language monitor or assembler. We would alienate a large number of readers if we assumed that they had certain software and wrote all of our articles accordingly.

Entering a program as decimal numbers contained within DATA statements isn't so bad, but it really is a waste of space and time to constantly use the program in this form. Each number in the DATA statements represents only one byte of machine code, so a BASIC loader typically takes up about five times as much memory as the machine language program itself, which is put into memory somewhere when the BASIC loader is RUN. Running the loader, especially for a big program, can also take a considerable length of time. Another problem with having an ML program in BASIC form is the fact that many ML subroutines are designed to be used from an existing BASIC program, meaning that the loader has to be merged, or loaded and run as an overlay, wasting more time yet.

Ideally, once a machine language program has been entered, it will exist in pure machine language form on disk or tape. Such a file can just be LOADed into its appropriate memory address and executed with a SYS command. The advantages of such a program:

1) It takes up much less space on disk or tape than a BASIC loader
2) It can be LOADed and not interfere with a BASIC program currently in memory
3) There is no waiting for a BASIC loader to put the program into memory

Using pure ML files on disk or tape, you can have a BASIC program which loads an ML program, executes it, then loads another one, etc. It is clearly desirable to put your BASIC loaders into pure ML form.

There are two ways to create an ML program file from a BASIC loader: RUN the loader, then save the resultant ML program from memory, or use the loader to write directly to a disk (or tape) file.

Eliminating the Loader, Method #1

As an example, let's use the "Quake" program from this issue's Bits & Pieces section. It is listed in loader form (with verifizer codes to cut down the entry errors). To create a pure ML "Quake" program, we can just save it from memory after the loader is RUN. From line 30 of the loader, we can see that the ML program occupies addresses 49152 through 49342. All we have to do is save that range to a program file on disk. That can be accomplished with an ML monitor, or for the C64/VIC we can use the technique given in Volume 5 Issue 5's Bits & Pieces section to save a range of memory from BASIC:

```
sys57812"0:quake.ml",8:poke193,0:poke194,192
:poke174,190:poke175,192:sys62954
```

(The start address, $C000 goes into 193–194 and the end address, $C0BE into 174–175.)

The program "quake.ml" will now be on disk, and can be loaded with the non–relocating LOAD command and executed:

```
load"quake.ml",8,1
sys 49152
```

The SYS address is determined from the loader program, where it is usually given in a REM statement or executed after POKEing in the DATA values.

Method #2

The second method forces the loader to write the ML program directly to disk, instead of putting it into memory. This is superior to method #1 because it doesn't interfere with any program that might be in memory when the loader is run.

The BASIC loader has to be modified slightly. The first thing to do is to add a command at the beginning of the loader which OPENs the file. Take the "Quake" loader, and add this line:

```
16 open 1,8,1,"0:quake.ml"
```

(You *must* use a secondary address of 1 unless the suffix ' ,p,w ' is added to the filename.)

The next step is to add a statement to the loader which will write the program's LOAD address to the file. This will instruct the LOAD command where in memory to put the program. The LOAD address is the start value of the FOR. . .NEXT loop which puts the program into memory. Looking at line 30 in the "Quake" program, we see that the start address of the ML program is 49152. This value must be written as the first two bytes in the now-open program file, in low, high format. Use the standard formula:

```
hi=49152/256
lo=49152-lo*256
(lo=0, hi=192)
```

For this example, then, we add the line:

```
17 print #1,chr$(0)chr$(192);
```

Now, the main modification: change the loader so that it PRINTs to the above file instead of POKEing to memory. To do this, replace the statement in line 30: 'POKE I,A' with: 'PRINT#1,CHR$(A);'.

The final step is to add a CLOSE statement after all writing to disk (this is crucial):

```
45 CLOSE 1
```

Once the loader has been modified in this way, RUN it and wait while it writes the file. After it's finished (assuming no DATA errors), you'll have the file "quake.ml" on disk, and the loader will never be needed again. You'll probably want to keep it, though, as a backup.

Method #2 Summary

1) Add to start of loader:

```
open 1,8,1,"0:prog name"
print#1,chr$(lo)chr$(hi);
```

Where lo,hi are the program start address
2) Replace 'POKE I,A' with 'PRINT#1,CHR$(A);'
3) Add after FOR. . .NEXT loop:

```
CLOSE 1
```

LOADing the ML Program

Your newly-created ML program can't be simply LOADed and RUN like the loader. You have to use the non-relocating LOAD command to place the program at its proper load address, then you have to execute the program with a SYS (or possibly a USR) command:

```
load "quake.ml",8,1
sys 49152
```

There is a slight complication when LOADing an ML program of this nature. BASIC sets it start and end program pointers after a direct-mode LOAD, so they will be messed up after you LOAD the ML program. You will not be able to edit your BASIC program after the load. The easy but possibly undesirable method is to issue a NEW after loading the ML program. Alternatively, if you are using the programmer's utility package 'POWER', a FIX or PTR command will fix up the pointers for you.

The other solution is to LOAD the ML program from an executing BASIC program. When a LOAD is encountered in a program, it does not change the pointers. It does re-run the BASIC program, though. If you want to LOAD an ML program from somewhere within the depths of a BASIC program, then continue execution, you can take advantage of the fact that variables aren't destroyed by the auto-run. For example:

```
10 on a goto 30,60,180
20 a=1: load "ml prg1",8,1
30 . . .
40 . . .
50 a=2: load "ml prg2",8,1
60 . . .
   .
   .
   .
170 a=3: load "ml prg3",8,1
180 . . .
    .
    .
    .
```

By choosing your favorite method from the above, you can transform all of your ungainly loader programs into efficient, ready-to-use machine language routines. Just one more way to conserve the precious commodities of time and space.

U What ?

Jesse Knight
Brazoria, TX

It's a case of mutual confusion. . .

If you wanted to send a reset command to the 1541, you might use code like this:

```
OPEN 1,8,15: PRINT#1, " UJ " : CLOSE 1
```

That seems simple enough; open the command channel, send the command UJ, close the command channel. It would be simple, and it would work fine, if it weren't for the CLOSE 1 at the end.

When the 1541 receives the UJ command, it begins the full reset sequence. This involves the testing of its 16K of ROM and 2K of RAM. While it is doing this, it ignores everthing else. The CLOSE command, in the example above, complicates things because it causes the computer the send a command byte to the drive. The computer tries to send the byte to the drive, but it encounters a problem.

To send the byte to the drive, the computer first sends an ATTENTION signal over the serial bus. Then the computer looks for an ATTENTION ACKNOWLEDGE signal from the drive. In this case, it receives one. Not because the drive actually sent it, at this point the drive is still busy with the ROM/RAM tests, but because of the value in the data port for the serial bus by the drive. The next thing the computer does is wait for another signal, called READY FOR DATA, from the drive. It will wait for this signal forever.

Normally this is an important and useful part of the serial bus I/O. If the drive happened to be busy formatting a disk and it received the ATTENTION signal, it would respond with the ATTENTION ACKNOWLEDGE signal. Then it would finish formatting the disk. After it finished formatting the disk, it would send the READY FOR DATA signal and the communications could continue. If the drive didn't send the ATTENTION ACKNOWLEDGE signal, the computer would generate a DEVICE NOT PRESENT error.

That's what might happen normally, but the problem is what happens during the reset sequence. The computer has received the ATTENTION ACKNOWLEDGE and is waiting for the READY FOR DATA signal. Once the drive finishes the reset sequence, it waits for something to do. It doesn't know it's supposed to be listening to the computer. It's a case of mutual confusion with the drive waiting on the computer and the computer waiting on the drive.

There are two solutions to this problem. The first should be rather obvious. Use a delay after sending the UJ command before attempting to access the drive. I have found a FOR NEXT loop from 1 to 1000 will work fine. The second solution is to use the alternate reset command, UI. The UI command performs a reset but it skips the ROM/RAM tests. This makes two differences. First, it takes less time. Secondly, most of the 2K of RAM is not altered, whereas UJ sets it all to zero.

There are actually two things the UI command can be used for. UI alone causes the reset skipping the ROM/RAM test. The second is to set a timing value the 1541 uses for the serial I/O. UI+ causes it to be set to work with the 64, while UI- causes it to be set to work with the VIC-20. It will work with the VIC-20 with either setting, but using UI- gives a slight increase in data transmission rate. The longer delay is needed with the 64 because of the VIC II chip.

The program in listing 1 will allow you to see part of the serial bus I/O. The program copies the BASIC and KERNAL ROMs to RAM and makes a patch to the routine used to send data over the bus. This patch causes the screen border color to be incremented each time the bus is checked for the READY FOR DATA signal. When executed, the program uses three methods to reset the 1541, so you can see how they work. When it gets to the last one, UJ without any delay, the computer will appear to lock up. The border will be a mass of colored lines. Press the RUN/STOP and RESTORE keys to regain control.

At this point the command POKE 1,53 will switch to the modified KERNAL in RAM. Now each time the computer sends data to the drive the border color will change. Try loading and saving programs or data files and watch what happens. Something else you may want to try is this: use something like:

```
OPEN 15,8,15, " N0:EXAMPLE,XX "
```

to start a disk format. While the format is in progress, give the command:

```
LOAD " $ " ,8
```

Watch what happens now, and when the disk format is finished.

Who knows, you might have a flashing good time.

```
BM  100 rem visual uj   jk 8/23/85
KO  110 print chr$(147)" copying rom to ram. . ."
JK  120 print " takes about one minute"
JA  130 for x=40960 to 49151: poke x,peek(x): next
OA  140 for x=57344 to 65535: poke x,peek(x): next
AA  150 for x=0 to 5: read a: poke 49152+x,a: next
OG  160 poke 60763,0: poke 60764,192
GL  170 poke 1,53
DI  180 print: print " press space to continue"
NG  190 get a$: if a$<>" [1 space] " then 190
CL  200 print: print: print " uj using delay. "
KE  210 open 1,8,15
BI  220 print#1," uj "
KM  230 for x=1 to 1000: next
DJ  240 input#1,en$,em$,et$,es$
GN  250 print en$,em$
HN  260 close 1
NN  270 print: print " press space to continue"
GM  280 get a$: if a$<>" [1 space] " then 280
EL  290 print: print: print " ui without delay. "
EK  300 open 1,8,15
IN  310 print#1," ui "
DO  320 input#1,en$,em$,et$,es$
GC  330 print en$,em$
HC  340 close 1
NC  350 print: print " press space to continue"
CB  360 get a$: if a$<>" [1 space] " then 360
BF  370 print: print " uj without delay. "
EP  380 open 1,8,15
LC  390 print#1," uj "
DD  400 input#1,en$,em$,et$,es$
GH  410 print en$,em$
HH  420 close 1
LC  430 data 238, 32, 208, 76, 169, 238
```

1571 Notes

The 1571 drive is a very capable unit. Commodore promised that this drive would be totally compatable with the 1541, and it really appears to be.

It is also intelligent. When connected to the 128 computer, it works in 1571 mode. If it happens to be connected to a 64, it works like a 1541. That is done automatically.

There are added commands that allow the modes and features to be controlled through software as well. To help maintain compatability with the 1541, these commands were added as part of the U0 command. On the 1541, the U0 command was only used to reset the pointer to the user command jump table. On the 1571 it is used for much more.

The command U0>M0 places the drive in 1541 mode. In this mode it acts just like a 1541.

The command U0>M1 places the drive in 1571 mode. In this mode the fast serial bus routines can be used. Disks are double sided by default in this mode. That means 1328 blocks free instead of just 664. The back side of the disk is formatted with tracks 36 through 70. This is treated the same way as with the 8050 and 8250 drives. The single sided disks can be read in the double sided mode and the front side of double sided disks can be read in single sided mode.

If you think that's confusing, wait until you read about the next command. The commands U0>H0 and U0>H1 are used to select which head to use. This command only works when the drive is in 1541 mode. This allows each side of a disk to be treated as a separate disk, with tracks 1 through 35 on each side. It's not the same as cutting an extra notch in a disk to use the back side, since the rotational direction is different.

The command "U0>R" +chr$(X) sets the DOS variable REVCNT to the value of X. REVCNT is used to control the number of tries made to recover from a read error. The upper bit is used to control the the head 'bump'. The number of retries can be set to 10 and the head bumping disabled by using "U0>R" +chr$(138).

The command "U0>S" +chr$(X) sets the sector interleave to the value of X. The normal interleave is 10.

There is even a command for setting the device number. "U0>" +chr$(X) sets the device number to the value of X. It may be any value from 4 through 30.

The device number can still be changed through hardware. Fortunately, the case doesn't have to be opened to do this. There are two DIP switches on the back of the drive to use. Device numbers from 8 to 11 can be selected this way.

There are more commands, but they are mostly for the MFM mode. One command lets you determine the disk format (MFM or GCR). Another is for formatting a disk in MFM format. Even in MFM mode the 1571 is flexible. It can handle tracks with 128, 256, 512, or 1024 byte sectors.

There's no doubt that the 1571 is a capable unit. It will work fine with disks and software for the 1541. The new commands and features make it a really nice drive. It's at least enough to keep a person busy.

Solving SAVE@: "So Close We Can Taste It!"

Karl J.H. Hildon

If you've been following the SAVE@ scene, then you've probably read by now the article in Compute by Phillip A. Slaymaker. Mr. Slaymaker has presented some truly significant information. His program demonstrates the bug in any 1541 thus proving that nobody is immune.

Slaymaker's program creates a disk that will show a failure to allocate sectors that are used by a PRG file. Once the disk is set up, he specifies a LOAD followed by three SAVE@'s. On the third SAVE@ the Blocks Free count is 4 greater than it should be. However, if the LOAD command is entered with a drive number 0 included, the third SAVE@ does not fail.

Using this information along with the veritable encyclopedia of other data we've collected, we managed to produce some rather interesting results.

First of all, Slaymaker states in the article that, "the key to avoiding the SAVE@ bug is to always specify drive 0 when performing any disk drive function, or to always reset the drive before any SAVE@ operation.". A 'UJ' command is suggested for resetting the drive, but as you'll note in the previous article ("U What?" page 68) there are certain pitfalls to be aware of. In a telephone conversation with Phillip Slaymaker I learned that he too is aware of the UJ problems but, understandably, including them was beyond the scope of his article.

Before we even entered the Slaymaker demo program, we tested the drive 0 theory with the Whittern SAVE@ loop we published some issues back. Whittern's program omits the drive number in all three LOAD commands. The program was modified to include "0:" in each. Then, using a newly formatted disk, 5 programs consisting only of REM statements, all equal in size, were SAVEd to the disk, followed by the modified Whittern test. The drive was reset with a power off/on and the test program was LOADed with drive 0 prefixed to the filename.

Guess what. No sooner had the second SAVE@ been performed when I got the urge to hit the STOP key. A quick look at the files showed the program selected by the second SAVE@ had overwritten the program selected in the first SAVE@. Inotherwords, the first SAVE@ failed to allocate sectors, and the second SAVE@ came along and clobbered them. As it turned out, the program that got clobbered was "program 1" – the first in the directory and the first to be written to the fresh diskette.

The 5 programs stored were short 2 block files. All 5 programs plus the test program fit comfortably on one track. Slaymaker claims that files spanning more than one track are more susceptible. Further tests showed this claim to be true, however it seems that no file is absolutely safe. Further in our telephone conversation we also agreed on this.

Although P.A. Slaymaker has planted an impressive milestone, there are still some unexplained phenomena. For example, the block allocate failures are not limited to SAVE@. A Scratch followed by a SAVE in place of SAVE@ in the Whittern test will also scramble files. Secondly, the Slaymaker demo was re-cut for the 4040 drive but failed to show any problem. Based on the number of SAVE@ reports prior to the 1540/41, recent developments, AND personal experience, it is highly doubtful that the other CBM drives are failsafe (and I could probably find at least a few people who would agree). Mr. Slaymaker has indeed discovered an important flaw, but it may not be the only one! In our conversation we agreed on this too, however, the Slaymaker demo is so far the closest most finite test there is for solving this mystery.

Using Slaymaker's approach, we decided to make new test programs that would monitor the internal activities of the DOS. The fact that 1541 DOS is a descendent of the dual drive DOS is clearly explained in Slaymaker's article and clearly evident from looking at 1541 DOS code. We all know the 1541 has only one drive, but under certain predictable conditions the DOS prepares itself to handle two.

Before I continue, I'd like to point out that this is probably the most difficult explanation I've ever attempted. Should you notice some repetition, it's purely in the interest of clarity.

When no drive number is specified in commands sent to a dual drive, the DOS will activate both drives (if necessary). For example, Initialize without a drive number will initialize both drives. Same with Validate, and a LOAD will access both drives before reporting File Not Found. Even though the 1541 only has one drive, the DOS will allow for the "presence" of drive 1. That is, a LOAD with no drive number may find the file on drive 0, but DOS allows for the possibility that drive 1 may need service. This possibility means that work space must be set aside in disk RAM to service drive 1 for things like the drive 1 Block Availability Map (BAM). Remember, this activity is the result of residue left over from dual drive DOS.

The 1541 DOS uses five pages of disk RAM as "buffers". A buffer is 256 bytes long and always begins at a page boundary. The first buffer is at $0300 to $03FF. The second buffer is at $0400, and the fifth buffer starts at $0700. Two other buffers, the Command Buffer and the Error Buffer, are situated in page $02 and treated as the sixth buffer. The DOS treats this buffer much like the others, but arranges for this space to be flagged as "always in use" so that no other buffer related activities can disturb it.

> Now at this point we are very close to publication and have conjured several theories about the internal workings of DOS. From here on we can only offer what we believe to be correct based on observations. We fully intend to continue our investigations and will probably have a more accurate story to tell by next issue.

Theoretically, the 1541 need only accommodate one BAM. Upon performing the very first disk operation, the drive 0 BAM is transferred from the diskette into the buffer at $0700 which is flagged as "in use", leaving four buffers unused. If the first operation implies there may be a second drive, the DOS activates another buffer to host the drive 1 BAM. This leaves three buffers unused.

Let's assume only one buffer has been allocated for the BAM of drive 0. During a write operation to the disk, the DOS needs to use buffers that are often occupied by the BAM. A very peculiar activity begins here called "floating BAMs". The DOS actually transfers the contents of the BAM buffer to other buffers which, in turn, become the new BAM buffer. As each block is pumped onto the diskette surface, the DOS must update the BAM with the new allocated sectors. Things are happening pretty quickly at this point (believe it or not) and the DOS doesn't have time to make all the calculations necessary to zoom in on the bit representing the sector it just wrote to. Instead, an "image" of part of the BAM is placed elsewhere in memory and updates occur here. Later the BAM is updated using the updated image. Let's expand on this.

Since writing occurs to the sectors of one track at a time, the BAM information for that track is transferred from the BAM buffer to a BAM image at $02A1 to $02B0. Each track requires four bytes to represent the free/used sectors of the track. The first byte tells how many free sectors on the current track, the remaining three bytes show which sectors they are. A bit set to zero means used, bit = 1 means free. Three bytes times eight bits equals 24 bits, which is enough for even the largest tracks at the outside edge of the diskette. The 16 bytes from $02A1 to $02B0 store track information for two tracks of drive 0 and two tracks of drive 1. The last eight bytes should never be used. (One theory we toyed with was the possibility of updating to or from the drive 1 image by mistake, but have since discounted that theory as drive 0 is specified in the SAVE@ commands)

As the write progresses, the image is adjusted for the sectors that have been written to. When all free sectors of the current track have been used up, a new track will be needed to continue. The DOS transfers the four-byte image back to the BAM buffer and it appears they will always be put back to the same four addresses they came from. As the BAM is updated the image is cleared with zeroes. So even if not all the sectors on the track were used, the image will now look as if they were. (This may be why some have reported diskettes that mysteriously "fill up" as opposed to too many Blocks Free - we intend to investigate that too!)

After the BAM is updated, the DOS searches out a new track with available sectors. Another tracks' worth of bytes are placed in the image, and the BAM buffer floats away again as the writing plows on. The BAM is only written to the diskette at the end of the operation.

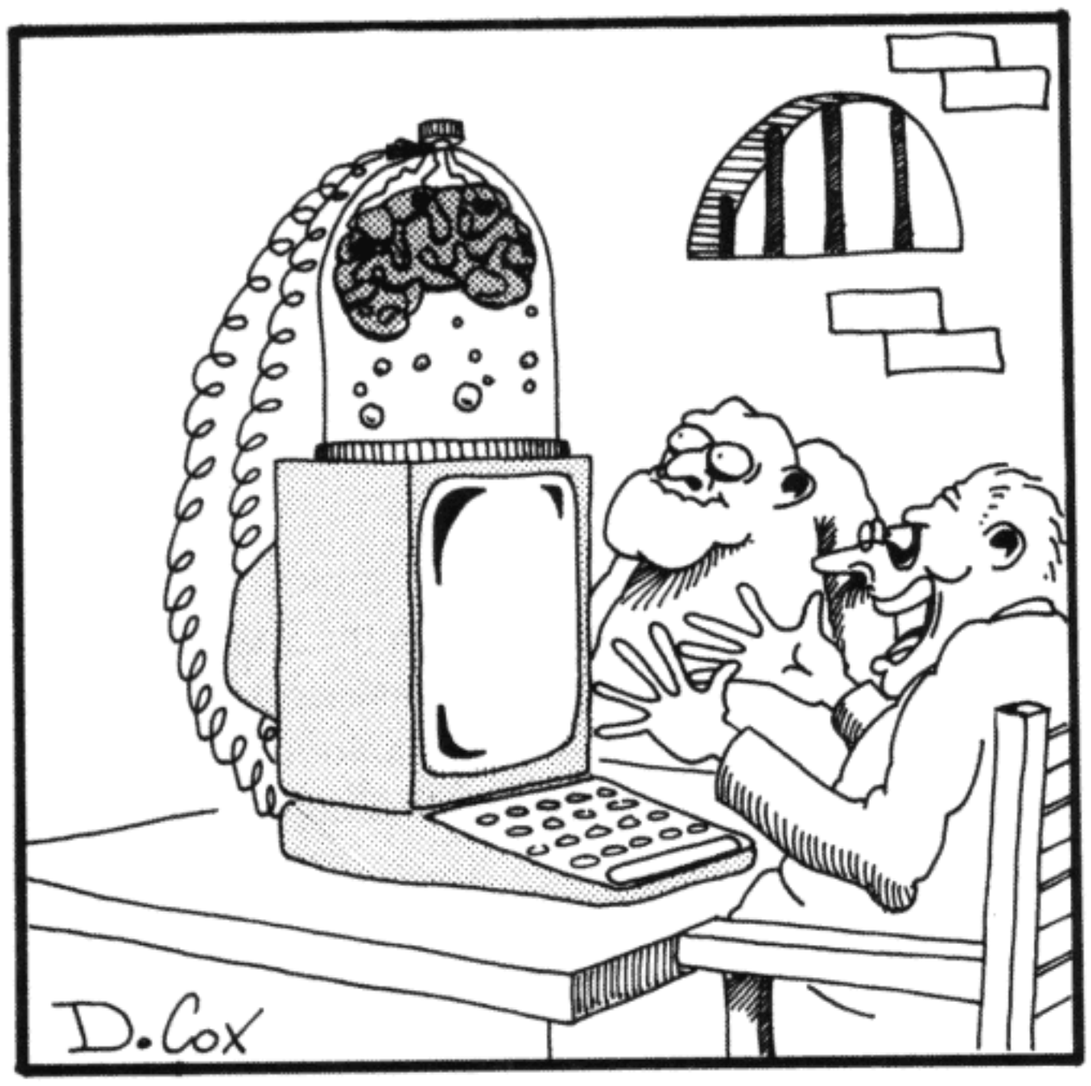
With only one buffer active as a BAM buffer, this operation seems to go without trouble, at least with the Slaymaker demo. But when two buffers are taken for BAM storage (remember way back when DOS allowed for the phantom drive?) the DOS is now burdened with the extra task of floating two BAM buffers instead of one.

During a write, the DOS is desperate for buffer space. Buffers that are active must be made inactive so the DOS can re-use them. This is done by a routine that "steals" buffers. In my conversation with P.A. Slaymaker, we agreed that the stealing routine could very possibly steal the buffer hosting the drive 0 BAM. Should this happen the BAM would disappear completely from disk RAM! Then another routine comes along and detects that nowhere in RAM is there a copy of the BAM. What happens now? The DOS re-reads the BAM from track 18, sector 0. But this BAM reflects the state of the diskette before the writing started. Any information that was swapped from the image into the BAM buffer would be lost forever! This is what happens during Slaymaker's demo.

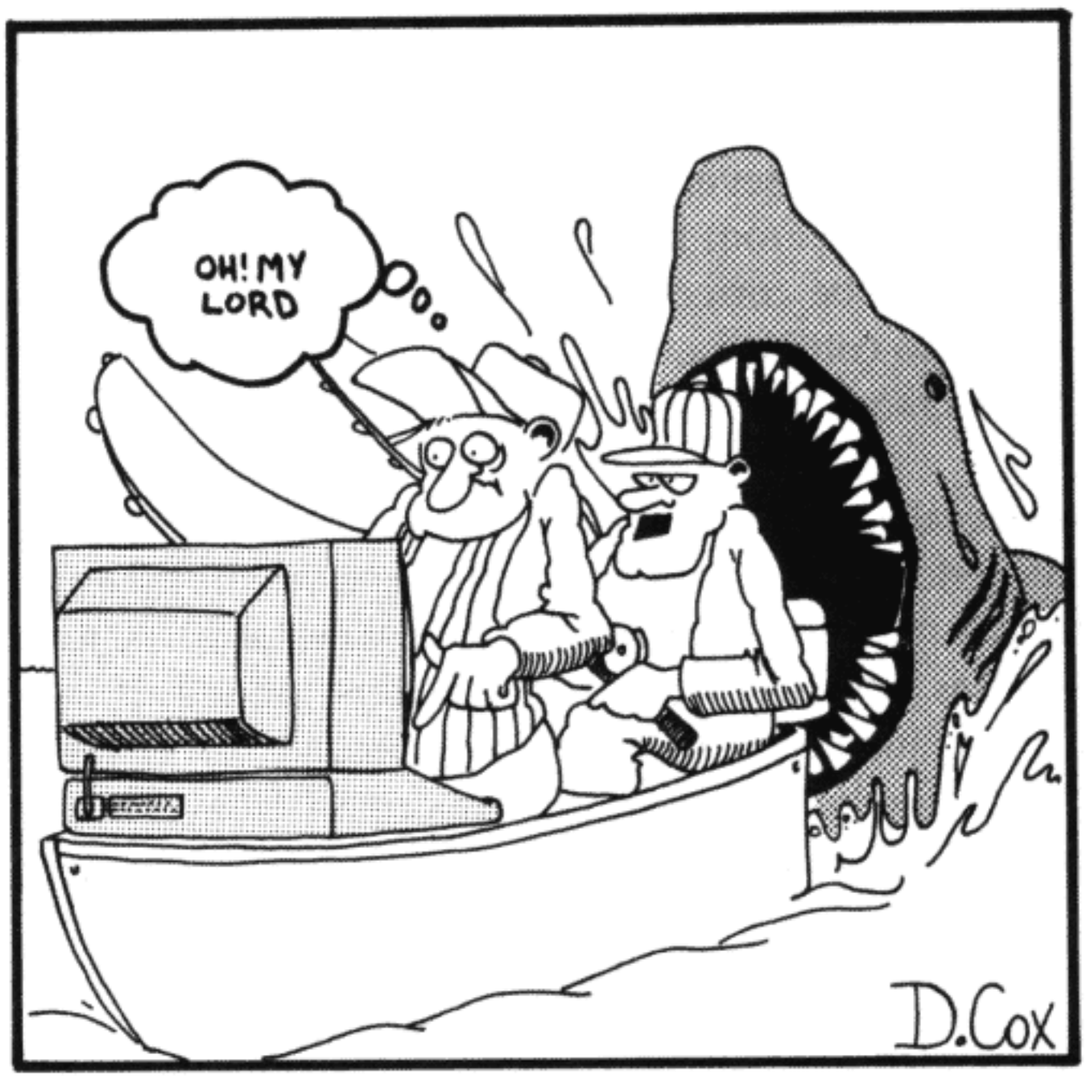
We've come up with several theories and if it weren't for a printing press that can't do anything without this page we may have had time to eliminate all but one. By next issue we should know:

1) Why the BAM is lost when two BAM buffers are activated, but also when only one is active.
2) Why disks are filling up due to incorrect image swaps.
3) Why SAVE@ fails on the 4040 which doesn't use the floating BAM concept.
4) The bytes to change in ROM to eliminate this bug.

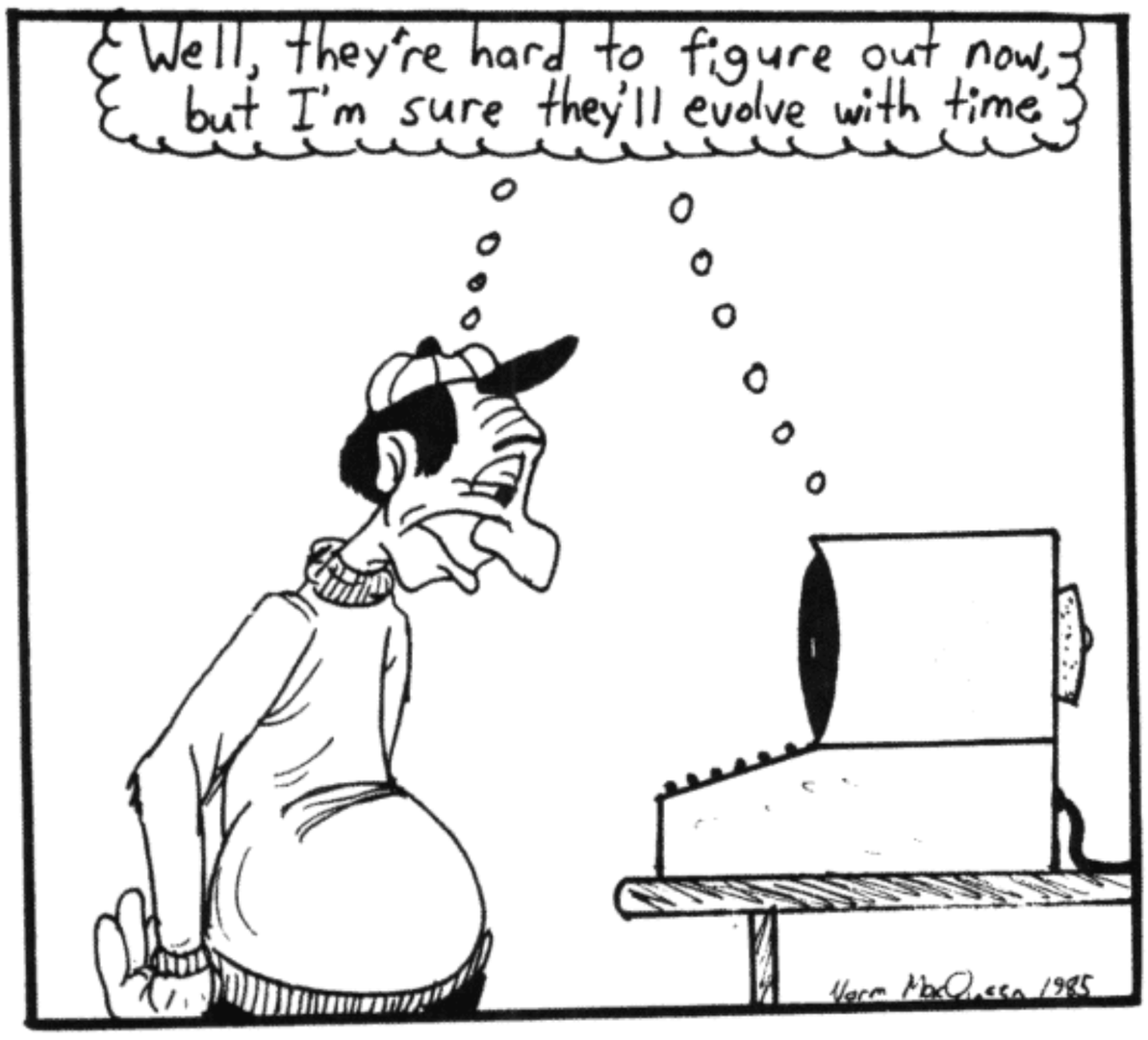
Compu–toons



HAAAAA! Egor. . . We've done it. . .
Expanded memory to 3 Billion K!



Forget it Ed. . . Your stupid computer isn't
gonna tell us where the fish are.



OK Ethel. . . One more POKE
and we're Instant Millionaires

CAPTAIN SYNTAX

I BELIEVE YOU MAY KNOW MY GUEST....

THE PRIME MINISTER?

I AM HIS SERVANT.

!

NOW MY LACKEY, GET CAPTAIN SYNTAX!!

OOOF!

I CAN'T HIT THE P.M.!

BUT I DON'T WANT TO BE KILLED!

GOT TO SUBDUE HIM...

BUT...

GOOD WORK, B.M.! NOW WHILE HE'S OUT, APPLY THE MIND CONTROL CHIP!!

TO BE CONTINUED???

News BRK

Transactor News

Submitting NEWS BRK Press Releases

If you have a press release which you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way applicable to Commodore equipment, with the exception of products or news of interest to the general computing public.

Oops, Too Many Labels

Did anyone receive two magazines last month? Or maybe you received a magazine even though your subscription had expired. The reason? When a subscription expires, our data base flags it as 'inactive' as opposed to deleting it completely. If that particular subscriber renews, only the flag need be changed which saves us the trouble of re–entering from scratch. After 6 months inactive, the record is discarded. When the labels are printed, the system looks at this flag. Except last issue it didn't. So instead of printing just the active subscribers, it dumped the entire data base. Those few who received two are probably in our data base twice; once as 'active', the other 'inactive'. Regardless, if you received a magazine you weren't expecting, please keep it with our compliments.

Late Note On Transactor Disk 7

Transactor Disk 7 for the Networking and Communications issue contains a set of terminal programs for the 64. You may have noticed some problems with these programs but the fix is easy:

```
open 1,8,15
print#1, "r0:firstterm3 bt=0:firstterm3 boot"
```

Renaming that one file will eliminate all but one problem: the program "extra extra" contains some brief instructions that refer to the file "firstterm3 boot", but only perfectionists will want to change that.

$4.50 Too Much

Transactor back issues are $4.50 each. If you send us $4.50 for a back issue we've just run out of, our policy was to send a refund of $4.50. But some U.S. readers have told us it costs them as much as $7.50 to cash the cheque. So unless you object, we'll add 2 issues to your subscription instead.

Commodore News

Commodore Introduces Technical Bulletin

Toronto — To provide the latest in technical information, Commodore Business Machines Limited is introducing TECHTOPICS - a bulletin program announcing modifications, troubleshooting and other technical topics concerning Commodore computers and peripherals.

The first seven issues in the series include: Troubleshooting tips for the 1702 monitor; specs and assembly upgrade for the 1541 disk drive; C64 PCB assembly update; C16 and +4 troubleshooting aides plus a listing of line definitions for the C16 and +4

Issues of TECHTOPICS are available upon request from the Support Department, Commodore Business Machines Limited. For further information:

Rainer Scharnke,
National Service Manager,
Commodore Business Machines
3370 Pharmacy Avenue
Agincourt, Ontario
M1W 2K4 (416) 499-4292

COMAL News

COMAL 0.14 Price Reduction

The power of Pascal, ease of BASIC, and fun of Logo turtle graphics can now be yours for only $7. This price includes the full COMAL 0.14 system as well as an interactive tutorial and automatic demonstration disk. It is not copy protected. In fact, you are encouraged to make copies!

Deluxe Cartridge Price Cut

Now, for a limited time, you can get the 64K COMAL 2.0 cartridge at almost $40 off the regular price. The DELUXE COMAL 2.0 CARTRIDGE PAK is now only $89.95 and includes a 320 page tutorial guide and 5 demonstration disks.

Price Protection Plan

If you buy any COMAL book, disk, or cartridge from COMAL Users Group, USA Limited, you are now protected for one month. If the price drops within that time, you are entitled to a credit of the difference. This protection is just one of the many benefits shared by COMAL TODAY readers. For information about this new service contact:

Denise Bernstein
COMAL Users Group, USA, Limited
6041 Monona Drive
Madison, WI 53716 (608) 222-4432

Product News

Toronto Computes!

Toronto Computes! is a monthly, mass-circulation paper keeping its readers up to date on the local microcomputer scene. 70,000 copies are distributed free through computer stores in the Toronto area and to homes in mid-town Toronto, a prime market area. The publication is geared towards all microcomputer users from the enthusiast to the novice. Toronto Computes! has information on the local scene that all computer users want to read - where to buy, available services, coming events, classifieds, innovative uses of computers in town and much more. Unlike the national and international computer magazines, Toronto Computes! is not packed with high-level technical information of interest mainly to the computer buff.

Cross Reference of Printers to Ribbons

Aspen Ribbons, Inc., of Lafayette, Colorado U.S.A., has recently published its 1985 catalog of "Ribbons for Computer Printers," and a new (first edition) "Cross Reference Guide."

The 40–page catalog of "Ribbons for Computer Printers" contains the photographs and names of 840 computer printer ribbons, complete with "How–to–Order" instructions and information on ribbon recycling and colors. It's attractive, easy to read, and FREE to anyone requesting a copy. Most of the ribbons listed are manufactured by Aspen Ribbons, Inc.

The new "Cross Reference Guide" contains over 8,000 ribbon–to–printer listings and over 1,500 computer ribbon model number listings arranged in a 2–color, 56–page booklet. For more information:

Aspen Ribbons, Inc.,
555 Aspen Ridge Drive
Lafayette, CO 80026 303 666–5750

New Commodore 128 Books

Commodore is once again drawing lots of attention with its new C-128 computer. And with virtually no competition in the low-cost home market, the C-128 will be a runaway success - just like its predecessor, the C-64.

So Abacus Software is announcing the first titles that will become part of a complete and in-depth reference library for the C-128. The initial titles and their availability are:

C-128 Internals -An inside look at the three computers inside the C-128. Includes ROM listings of BASIC 7.0 and operating system - Fall 1985
C-128 Tricks and Tips -collection of helpful techniques for anyone who programs the C-128 - Fall 1985
1571 Internals - An inside look at the brand new 1571 disk drive. Includes ROM listings - Winter 1985
CP/M on the C-128 -A closeup view of the CP/M operating system as found on the C-128 - Winter 1985
Artificial Intelligence -Intro to AI using the C-128 and C-64 - Fall 1985

Arnie Lee
2201 Kalamazoo S.E.
Grand Rapids, MI, 49510
(616) 241-5510, Telex 709-101

C Compiler for C64 & C128

Abacus Software announces the addition of two exclusive products – XPER and Super C Language Compiler for the Commodore home computer market.

XPER is the first expert system for the C64 and C128. Ordinary data base systems are good for reproducing facts, but by using an expert system you can derive knowledge from a mountain of facts and make expert decisions. Using this unique knowledge–based package, you first build the information into your data base using XPER's simple loading procedures. Then, by using very efficient searching techniques XPER can easily guide you through the most complex decision making criteria. XPER is currently used by scientists, doctors, and professionals in their research and studies. The XPER System includes full reporting and data maintenance capabilities.

The Super C Language Compiler is a complete development system. The powerful editor handles source files up to 41K in length. The fast compiler produces 6510 machine code. The linker accepts up to seven modules and the library supports standard as well as Commodore oriented functions. It conforms to the Kernighan & Ritchie standard.

3D Graphics System

Victoria, BC — In co-operation with Inkwell Systems, Pioneer Software Inc. announced the release of "FLEXI"AIDED DESIGN, a 3D graphics creation/manipulation/animation system for the C-64 and C-128.

The combination of "FLEXI"AIDED DESIGN and FLEXIDRAW used with Inkwell System's quality light-pen, provides the end-user with a complete graphics system, offering a price/performance ratio which has been completely unheard of until now.

Imagine being able to take a group of objects (we'll use a city as our example) and view it from any angle (overhead, far away, nearer, from the inside looking out). How about taking your city and enlarging it, reducing it, rotating it on different axes or doing any of these manipulations on a single object within the city, one of the buildings, perhaps. How about taking a group of views and creating an animation? All of the above and more is possible using "FLEXI"AIDED DESIGN and your C-64 or C-128. "FLEXI"AIDED DESIGN's operating system is destined to become a standard among Commodore users. Through the use of light-pen controlled, pull-down menus, windows, and graphics creation and manipulation, the user need never touch the keyboard once the computer is running.

SUPERSHIPPER 64

SuperShipper 64 is the complete invoicing and shipping system for your Commodore 64. All you need is a C–64, monitor, 1 dual disk drive OR two single drives, and 1 or more printers. SuperShipper FEATURES:

- Menu–Driven for ease of use.
- Flexible – virtually any combination of single or dual disk drives (including MSD) will work!
- High Capacity – 800 accounts per disk, 500 invoices & 200 products per disk. Expandable to 2,200 products with an additional disk drive.
- User–Friendly – disk and printer crashes are virtually nonexistent with SuperShipper's thorough checking.
- Easy Data Entry with machine language driven, full cursor controlled editing windows for entry of account, invoice and product data.
- Efficient Disk Use minimizes disk access time while taking full advantage of your disk drive's capacity.
- Protect Your System with two levels of access – Executive and Operator.
- Prints invoices of up to 31 lines on your choice of 8 1/2" X 11" paper or two types of NEBS pre–printed invoice forms.
- Prints C.O.D. tag with UPS Zone, UPS Shipper Number and user–selectable instructions.
- Prints mailing or shipping labels – features customized label formatting!
- Print alphabetical listings of accounts and products.
- Print lists of accounts and products sorted by data "keys" that you can specify!
- Prints a list of back orders.
- Invoices have provisions for Credit, Additional Charges, Tax and C.O.D. charges.
- Set up your own formula for shipping & handling charges.
- Four price categories for each individual product.
- Fixed or percentage commission breakdowns for each individual product.
- Set "default" values to streamline entry of account, invoice and product data.

Progressive Peripherals & Software
2186 South Holly,
Suite #2
Denver, CO 80222 303 759–5713

Music Printer For the C64

Sight & Sound Music Software, Inc., announces that in response to tremendous demand from users of its Music processor (a software program for musical composi-

tion), it has added a music printout feature to this program. The quality and accuracy of the printed notation ensure its recognition as the finest music print program now available for the Commodore 64.

The Music Processor now becomes the perfect program for musical composition, both for those with limited musical knowledge or for the more serious musician. With the new program, the user can compose a piece, then edit or make changes to it, then record the same piece and print out the final composition. All of these functions can be performed using up to three voices, thereby taking full advantage of the Commodore 64's unique three–voice SID chip.

To enhance an original composition, it's possible to make use of 99 preset instrumental sounds and special effects. Or for professional results with minimal effort, by using the joystick, the user can select from and change these 99 sounds and incorporate them into the 16 prerecorded songs that are part of the program.

Several other music software programs from Sight & Sound are compatible with the Music Processor. The Computer Song Albums make available dozens of contemporary hit tunes that the user can recreate by changing sounds and special effects. The Incredible Musical Keyboard fits over the Commodore keys and enables the user to play the Commodore keyboard just like a synthesizer. But the ultimate in creative expression is achieved when combining the Music Processor with the Music Video Kit. Now it's possible to design, orchestrate and record computer–animated music videos on the personal computer.

The new, upgraded Music Processor with printout feature retails for $29.95. Owners of the original can upgrade to the complete new program for just $15.00.

To upgrade, simply send proof of ownership and a check or money order made out to Sight & Sound Music Software, Inc. Ownership may be proven by either returning the original disk or cutting out the UPC and ISBN product numbers from the packaging. Please allow two to four weeks for delivery. For additional information, contact:

Jane Billings
Sight & Sound Music Software, Inc.
3200 South 166th Street
P.O. Box 27, Department R2D2
New Berlin, WI 53151 414 784–5850

SFD–1001 One Megabyte Disk Drive

The SFD–1001 (Super Fast Drive) is now available. With double–sided double–density format, over One Megabyte can be stored on a single floppy disk. One hundred 1541–formatted disks can be reduced to only sixteen SFD–1001–formatted disks. By using the intelligent IEEE bus and a bus expansion IEEE interface, the SFD–1001 loads programs and data over twice as fast as the 1541 drive, and all this inside a case the size of the 1541's!

Fully compatible with any Commodore computer that has an IEEE interface. Free utility disks for both the CBM 8032 and the C–64 are included! Transfer all your files and programs easily from any Commodore disk drive to your SFD–1001!

The SFD–1001 is available now from Progressive Peripherals & Software, Inc., your quality Commodore software and hardware source. Suggested retail price is only $399.95. Dealers inquiries are invited. . .call for more information or for the name of the dealer nearest you.

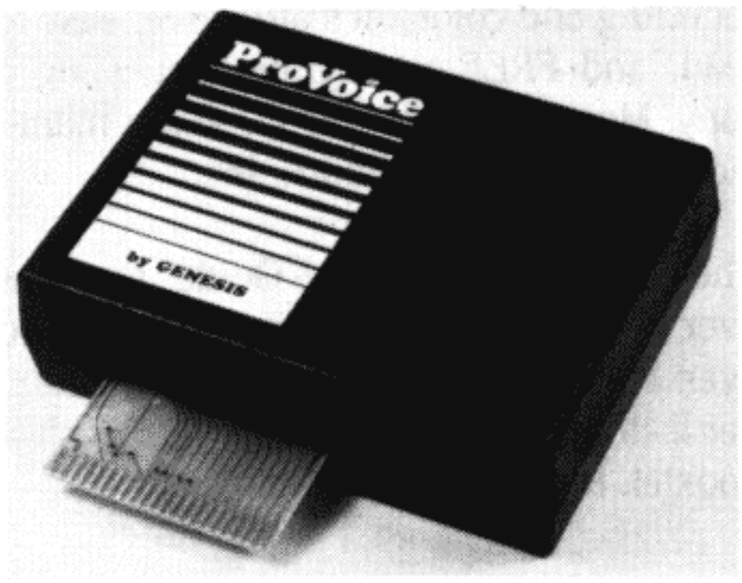
The 1541 loads 32K bytes of data in approximately 1 minute, 20 seconds, the SFD–1001 loads 32K bytes of data in only about 35 seconds (bus expansion interface) or approximately 1 minute, 4 seconds (serial interface).

Progressive Peripherals & Software, Inc.
2186 South Holly,
Suite 200
Denver, CO 80222 303 759–5713

Provoice Speech Synthesizer

Bethlehem, PA — Genesis Computer Corp. of Bethlehem, PA announces ProVoice, the latest version of its highly successful COMvoice speech synthesizer, for the Commodore 64 and compatible computers. ProVoice speaks an unlimited English vocabulary and contains the most sophisticated text to speech translation ever introduced for the Commodore computers. ProVoice has unique features such as screen echoing, which allows virtually any BASIC program to become a talking program, and variable translation modes for conversational, verbatim and character by character speech output. The screen echoing feature makes ProVoice an ideal aid for the visually impaired.

ProVoice adds 13 new BASIC commands, including a HELP feature for quick reference. All BASIC commands and text-to-speech translation are handled by Pro-Voice's on-board ROM. ProVoice is a single plug-in device containing the ROM and speaker/amplifier.



ProVoice will retail for $99.95 US, and a Talking Terminal package with modem, soon to be available, will have a targeted retail price of under $150 US.

Genesis Computer Corp.,
Ben Franklin Technology Center, Lehigh University,
Bethlehem, PA 18015 (215) 861-0850

COMPUTEREYES Video Acquisition Systems

Digital Vision, Inc. announced the appointment of PHASE 4 DISTRIBUTORS INC. as Canadian Distributor for the COMPUTEREYES line of video acquisition systems for personal computers. Priced surprisingly low, COMPUTEREYES represents the first cost–effective means of capturing real–world images on the computers' high–resolution graphics display. A complete system including COMPUTEREYES and a high–quality video camera is also available at a very reasonable price.

COMPUTEREYES is an innovative slow–scan device that connects between the computer and any standard video source (video tape recorder, video camera, videodisk, etc.). Under simple software control, a b/w image is acquired in less than six seconds. A unique multi–scan mode also provides realistic grey–scale images.

The accompanying images are printer screen dumps of images acquired by the system.

Many of the applications of COMPUTEREYES are obvious. These include pattern recognition, security, quality control, spatial measurement, robotics and artificial intelligence, industrial controls, computer art, education, and entertainment. Other applications are bound to surface, once the product is in the hands of the creative members of the personal computer community.

Comprehensive software is provided with the system and includes: machine language image capture routines; a menu–driven executive that provides everything even first–time users need to capture images; image save–to–disk capability; and image packing and unpacking routines that save disk space and speed loading and saving. To encourage application development and promote ease of use, the software is not copy–protected. Optional software is also available at a nominal charge to support many of the popular graphics manipulation programs, such as Print Shop and the Koala Pad.

The COMPUTEREYES package includes interface module, complete easy–to–use software support on disk, owner's manual, and one year warranty. The system is currently available for the Commodore 64 and the Apple II series, with an Atari 800/800XL/65XE/130XE version available.

Phase 4 Distributors Inc.
7157 Fisher Road S.E.
Calgary, ALTA T2H 0W5 403 252–0911

Omnitronics Printmaster/+G

The PRINTMASTER/+G is a full featured parallel printer interface compatible with all Commodore computers which use the Commodore type serial bus (C64, Plus4, C128, etc.).

The PRINTMASTER/+G allows complete emulation of a Commodore 1525 or 801 printer, including full graphics and graphics characters. Several advanced graphics features are also selectible, such as enhanced graphics, double density, and reverse graphics. Prints 6 or 8 bit wide Commodore graphics characters. Graphics printing speed is 6 to 12 times faster than many other printer interfaces. Prints a line of 80 graphics characters in 4 seconds. The PRINTMASTER/+G is compatible with virtually all popular printers, including Epson, Gemini, Tally, Okidata, Banana, and NEC. The PRINTMASTER/+G comes with a disk containing Hi–res printing program, graphics screens, banner program, and program examples. Cassette port or external power supply.

Intellifeatures are the advanced capabilities of the PRINTMASTER/+G, many of which are not available on any other interface. LOAD"S",4,1 displays a complete printer interface status on your computer screen. LOAD"$",4,1 displays disk the directory without erasing a BASIC program. Secondary Address Lock, Margin and Page length settings, Single Page pausing, and more.

An optional PRINTMASTER 16K Buffer Expansion adds additional buffer memory for quickly freeing up your computer, plus a second expansion ROM which adds even more advanced capabilities to the PRINTMASTER/+G. LOAD"RENUM",4,1 renumbers a BASIC program. LOAD"OLD",4,1 recovers a NEW'd BASIC program. Even faster graphics printer. Prints a line of 80 graphics characters in 2 seconds. Dot graphics printing faster also. Design and upload your own character set. Many more features.

The PRINTMASTER/+G and the PRINTMASTER 16K BUFFER Expansion are available now for $119.95 and $89.95 respectively. Product reviews are desired. For a full brochure, or to order, contact:

Omnitronix, Inc.
P.O. Box 43
Mercer Is., WA 98040 206 236–2983

EPSON HomeWriter 10

Fully compatible with all software for the C64, the HomeWriter is capable of printing in Standard print, Expanded print, Reverse field print, Expanded Reverse field prints, not to mention graphics capabilities.

One of the unique features of the EPSON HomeWriter 10 is SelecType print mode selection from the front panel, allowing easy access to a wide variety of print modes such as:

- Near Letter Quality text for word processing
- Emphasized print
- Double Strike print
- Compressed print for spreadsheets, for example

The PIC (Printer Interface Cartridge) which is quickly and simply installed in the back of the printer allows the same printer to work on a variety of computers such as the C64, Apple IIc and Atari home computers.

Epson America, Inc.
2780 Lomita Blvd.
Torrance, CA 90505 213 539–9140

Remote Keyboard Conversion Kit

If you have a Remote Keyboard Conversion Kit – Here's what you have:

- A keyboard for your lap, lean back–relax
- A keyboard to pass around when playing games
- A keyboard not restricted in movement by 5 cables
- A computer with cable plugs facing you
- A computer you can change cabling and accessories easily
- A color keyed to match original unit

Friendly Systems, Inc.
1845 Range Street
Suite A
North Boulder, CO 80301

MITEY MO 300 Baud Modem

MITEY MO is alive and getting better all the time. The upgraded version of the MITEY MO, being marketed exclusively by Computer Devices International of San Leandro, CA, now includes the "Smart 64 plus 4" terminal software in its new and enhanced package.

There has been some confusion surrounding the MITEY MO since it's originator, USI, filed bankruptcy and liquidated in the fall of 1984. Computer Devices International, CDI, purchased the rights to the product line, and is continuing to honor the three year warranty. CDI is also providing technical support, as well as a 48 hour turn around for any units that may need repair.

Adding to the confusion is the fact that there were a few thousand of the original MITEY MO's, without the "Smart 64 plus 4" terminal software, that were already in distribution when USI filed bankruptcy. These units are being sold at below market prices

through some discount houses and distributors. These cheaper units should be be confused with the MITEY MO's that are being advertised and reviewed by computer magazines and critics.

Currently, the only authorized Canadian distributors of the new MITEY MO are T.C. Data in Montreal, and Phase 4 Distributors in Calgary.

Anyone who has purchased on of the original MITEY MO's, without the new software, can upgrade it by contacting CDI directly. They are selling the "Smart 64 plus 4" terminal software for $19.95 US $, plus $3.00 US$ for shipping.

Computer Devices International
1345–A2 Doolittle Drive
San Leandro, CA 94577 415 633–1899

Mobile Data Terminal

Motorola's Communications Sector announces its newest member of a family of "wireless" data terminals; the KDT 480 Mobile Data Terminal.

Now it is possible to take full advantage of computerized operations in a vehicle with the same computer access capability available in an office. Plus, Motorola's KDT 480 terminal can be used as a dedicated radio system, or it can be incorporated into an existing radio system.

Field personnel can access computer networks from remote locations, customer sites and even after routine business hours for maximized efficiency in a mobile environment.

Storage of up to 3,000 characters in RAM are dynamically allocated to meet users' varying message mix and length.

Featuring a text area of over 20 sq. in., the highly visible CRT display can accommodate up to 480 high resolution, easy to read, characters, formated into 12 lines of 40 characters each for easy viewing. (Two additional lines provide 80 characters of operational status information)

The terminal display was designed for varying light conditions of the vehicular environment, to assure visibility. . .even in direct sunlight!

An emergency indicator enables a driver to transmit an alert by pressing a function key or remotely mounted switch.

The compact keyboard features full sized typewriter style keys, color coded by function for easy operation.

To meet varying requirements, the KDT 480 terminal's modular design allows separate mounting of all components for truly customized installations.

Unique Radio/Data Communications System

Motorola's Communications Sector introduces an industry first, designed to extend data networks into the mobile environment previously identified with two–way radio. The KDT 800 Portable Data Communications System operates in the 800 MHz frequency range. Two–way digital radio replaces traditional telephone lines to provide real–time communications between people on the move and computers.

A key element in the Motorola system is the battery operated KDT Computer Terminal which contains an 800 MHz data radio, internal antenna, a telephone modem and intelligence in excess of many personal computers. The environmentally rugged unit weighs less than 28 oz. and measures 7.5 by 4 by 1.3 inches.

The portable terminal features a 2–line LCD display of 27 characters per line. The 59 position keyboard has full alphabetic capability in standard typewriter arrangement, programmable function keys, and a numeric calculator pad.

Memory capacity is expandable to 160K bytes of ROM and 80K modules, one of which is externally pluggable. Application software has access to external memory and peripheral devices via either serial or parallel I/O interfaces. The KDT portable terminal can accommodate 1 megabyte of physical address space.

A spectrally efficient system design can support more than one thousand portable terminal users on a single radio channel within a geographic area with average message traffic. The system operates at 4800 bps, over standard 25 KHz or 12.5 KHz channels. In areas without radio coverage the portable terminal communicates with a central data base by connecting the built–in 300 bps telephone modem to any telephone.

Motorola's radio data network design can be implemented in a campus/plant environment (local area network) and for city–wide usage (Metropolitan area network). Metropolitan area networks can be linked to provide nationwide coverage. Over two–way radio channels, the portable units send and receive messages through fixed transmitter/receiver stations optimized for data traffic. The NCP–2000 network control processor provides message coordination across the entire terminal system. It tracks location of the KDT data terminal user, directs messages between the computer network and the terminal user, and controls operation of the radio equipment. In its full configuration, the network control processor contains seventeen 68000 microprocessors and has sixty–four ports programmable for interface inward to host computers or outward to fixed transmitter/receiver stations. The system also contains full remote and self–diagnostic capabilities.

The electronic office can now be carried in a pocket in the form of a KDT Computer Terminal. To receive additional information, contact:

Pat Schod
Motorola, Inc.
Communications Sector
Public Relations Department
1301 E. Algonquin Road
Schaumburg, IL 60196 312 576–6612

SPECIAL OFFER
$7.00
(NORMALLY $19.95)

Seeing Is Believing

"I don't have enough time or space to list all the good points!" -- ***Noland Brown, MIDNITE SOFTWARE GAZETTE***

"This disk is fantastic!" -- ***Tom Lynch, THE USERS PORT***

"Why all the enthusiasm? Because **COMAL** is a composite of the best features of the most popular programming languages... the familiarity of BASIC commands with the structural programming environment of Pascal and the turtle graphics of Logo." -- ***Mark Brown, INFO 64***

"**COMAL** was just what I was looking for." -- ***Colin Thompson, RUN***

Seeing ***is*** Believing. Take a look at what **COMAL** has to offer:
the complete **COMAL 0.14 System** for Commodore 64™ includes the
Tutorial Disk* *(teaches you the fundamentals of COMAL)*, plus
the **Auto-Run DEMO Disk*** *(demonstrates 26 COMAL programs including games, graphics, sprites and sounds)*,
all for just **$7.00!**
You can add the reference book, *COMAL from A to Z*,
for just **$4.00 more**.
$7 or **$11** -- either way you're a winner!

"Everybody who gets it, likes it! (I'll guarantee it.)" -- ***Len Lindsay, President, COMAL Users Group***

For more information or to place an order call (608) 222-4432. Visa or Master Card accepted; checks or money orders must be in U.S. dollars.

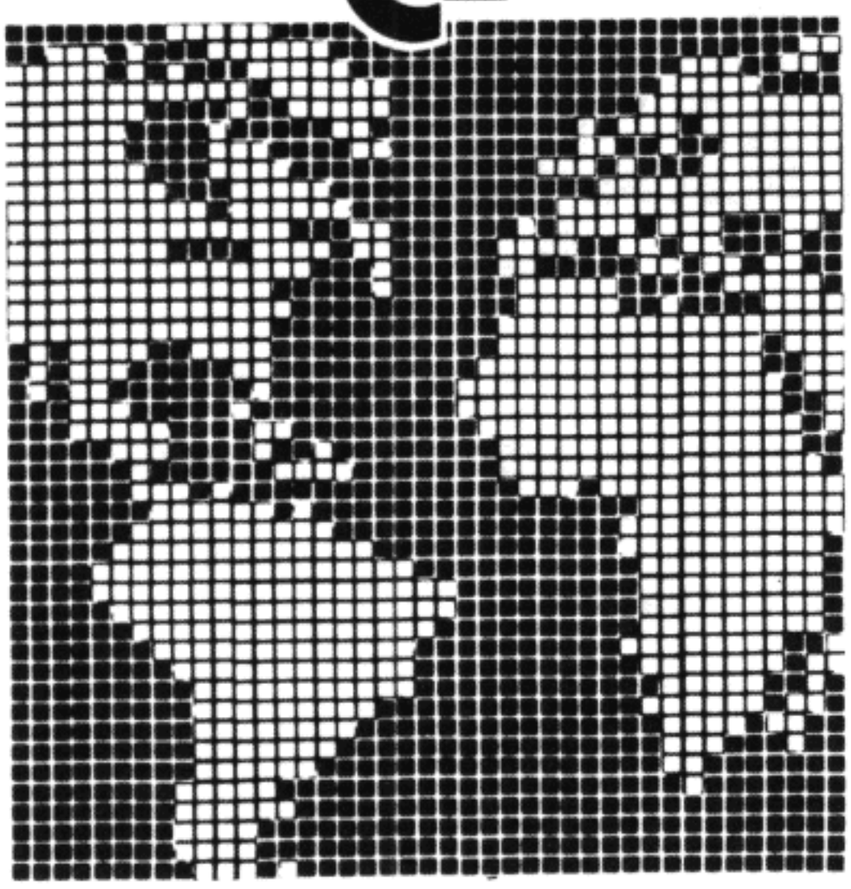
All orders pre-paid only - no C.O.D.

Send check or money order in US Dollars to:

COMAL USERS GROUP U.S.A.. LIMITED
6041 Monona Drive, #110, Madison, WI 53716
phone: (608) 222-4432

***Programs will come on 2 disks or 1 double sided disk -- each disk includes COMAL.**
Commodore 64 is a trademark of Commodore Electronics

THE WORLD OF COMMODORE III



The 1984 Canadian World of Commodore show was the largest and best attended show in Commodore International's history. Larger than any other Commodore show in the World and this year's show will be even larger.

World of Commodore III is designed specifically to appeal to the interests and needs of present and potential Commodore owners.

Everything about your present or future Commodore computer – from hardware to software, Business to Personal to Educational – from over 90 International Exhibitors. Price of admission includes free seminars, clinics, contests and free parking.

INTERNATIONAL CENTRE
DECEMBER 5 TO 8/85
TORONTO

A HUNTER NICHOLS PRESENTATION
For more information call:
(416) 439-4140

JOIN TPUG

The largest Commodore Users Group

Benefit from:

Access to library of public domain software
for C-64, VIC 20 and PET/CBM

Magazine (10 per year) with advice from

Jim Butterfield
Brad Bjomdahl
Liz Deal

TPUG yearly memberships:

Regular member (attends meetings)	—$35.00 Cdn.
Student member (full-time, attends meetings)	—$25.00 Cdn.
Associate (Canada)	—$25.00 Cdn.
Associate (U.S.A.)	—$25.00 U.S.
	—$30.00 Cdn.
Associate (Overseas — sea mail)	—$35.00 U.S.
Associate (Overseas — air mail)	—$45.00 U.S.

FOR FURTHER INFORMATION:
Send $1.00 for an information catalogue
(tell us which machine you use!)

To: TPUG INC.
DEPT. A,
1912A AVENUE RD., SUITE 1
TORONTO, ONTARIO
CANADA M5M 4A1

COMAL INFO
If you have COMAL—
We have INFORMATION.

BOOKS:

- COMAL From A To Z, $6.95
- COMAL Workbook, $6.95
- Commodore 64 Graphics With COMAL, $14.95
- COMAL Handbook, $18.95
- Beginning COMAL, $22.95
- Structured Programming With COMAL, $26.95
- Foundations With COMAL, $19.95
- Cartridge Graphics and Sound, $9.95
- Captain COMAL Gets Organized, $19.95
- Graphics Primer, $19.95
- COMAL 2.0 Packages, $19.95
- Library of Functions and Procedures, $19.95

OTHER:

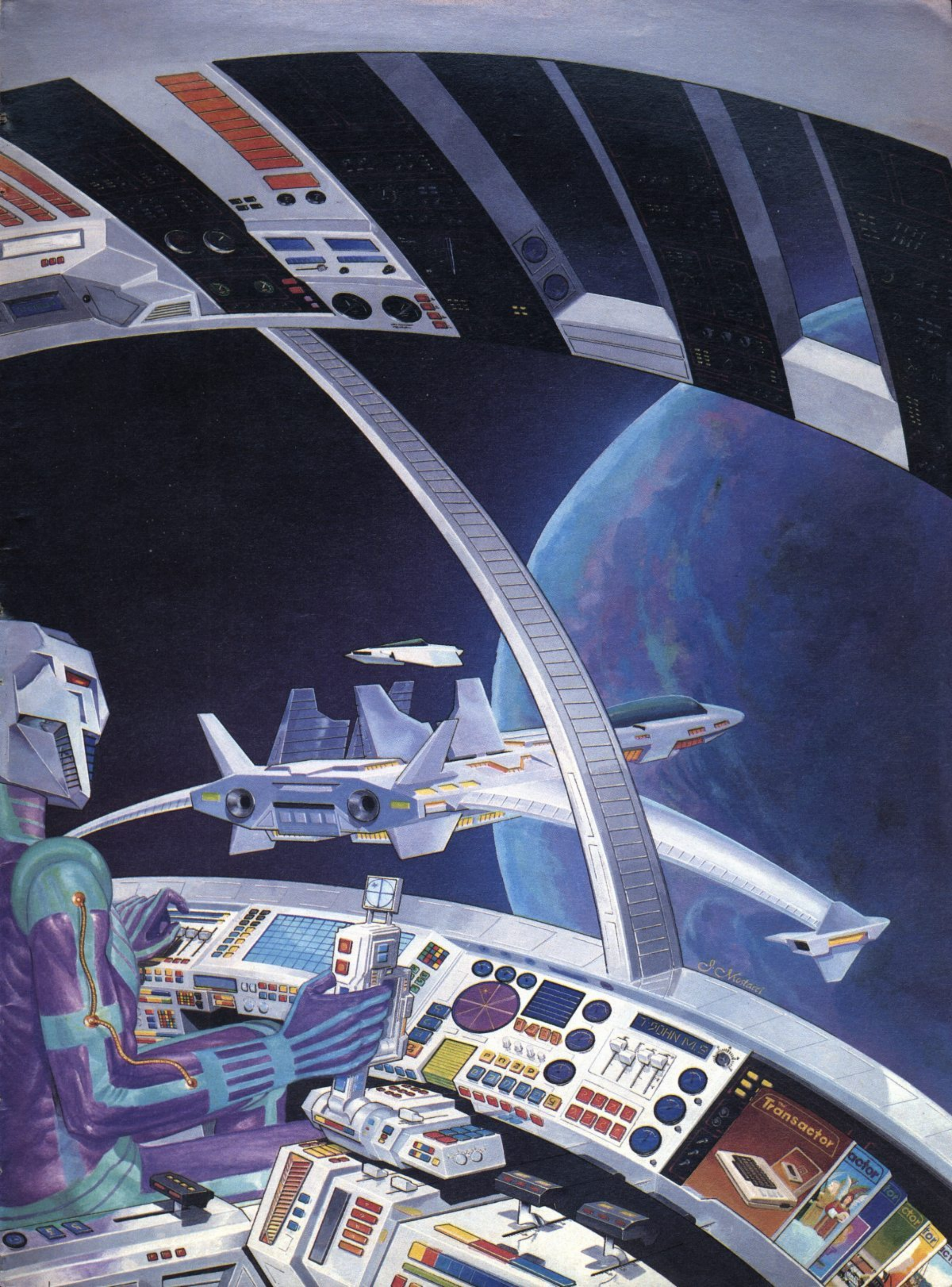
- COMAL TODAY subscription, 6 issues, $14.95
- COMAL 0.14, Cheatsheet Keyboard Overlay, $3.95
- COMAL Starter Kit (3 disks, 1 book), $29.95
- 19 Different COMAL Disks only $94.05
- Deluxe COMAL Cartridge Package, $128.95 (includes 2 books, 2 disks, and cartridge)

ORDER NOW:

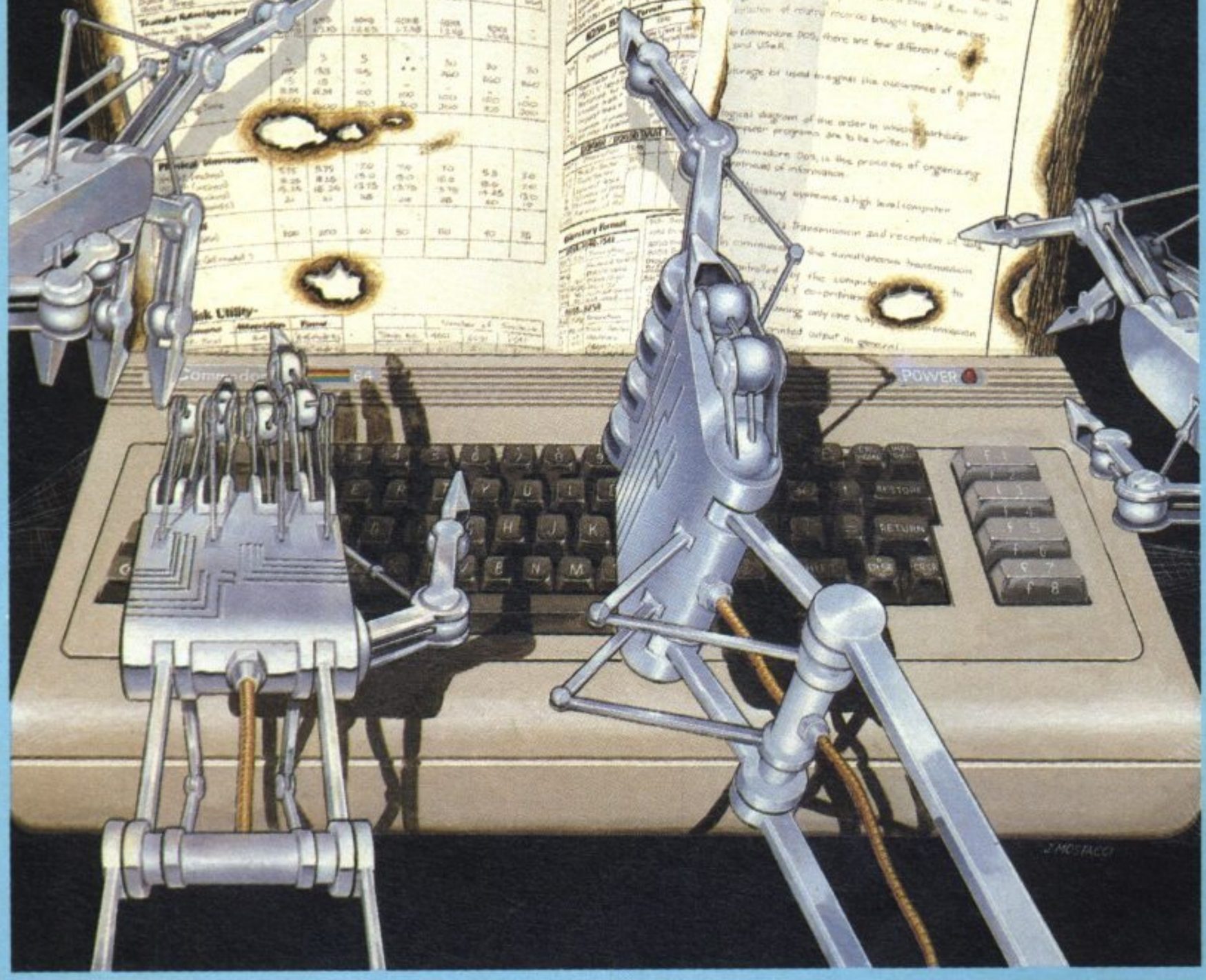
Call TOLL-FREE: 1-800-356-5324 ext 1307 VISA or MasterCard ORDERS ONLY. Questions and Information must call our Info Line: 608-222-4432. All orders prepaid only—no C.O.D. Add $2 per book shipping. Send a SASE for FREE Info Package or send check or money order in US Dollars to:

COMAL USERS GROUP, U.S.A., LIMITED
5501 Groveland Ter., Madison, WI 53716

TRADEMARKS: Commodore 64 of Commodore Electronics Ltd.; Captain COMAL of COMAL Users Group, U.S.A., Ltd.


T-JOHN MS
The Transactor
J. Mostacci

The Transactor presents, The Complete Commodore Inner Space Anthology



Over 7,000 Delivered Since March '85

Postage Paid Order Form at Center Page